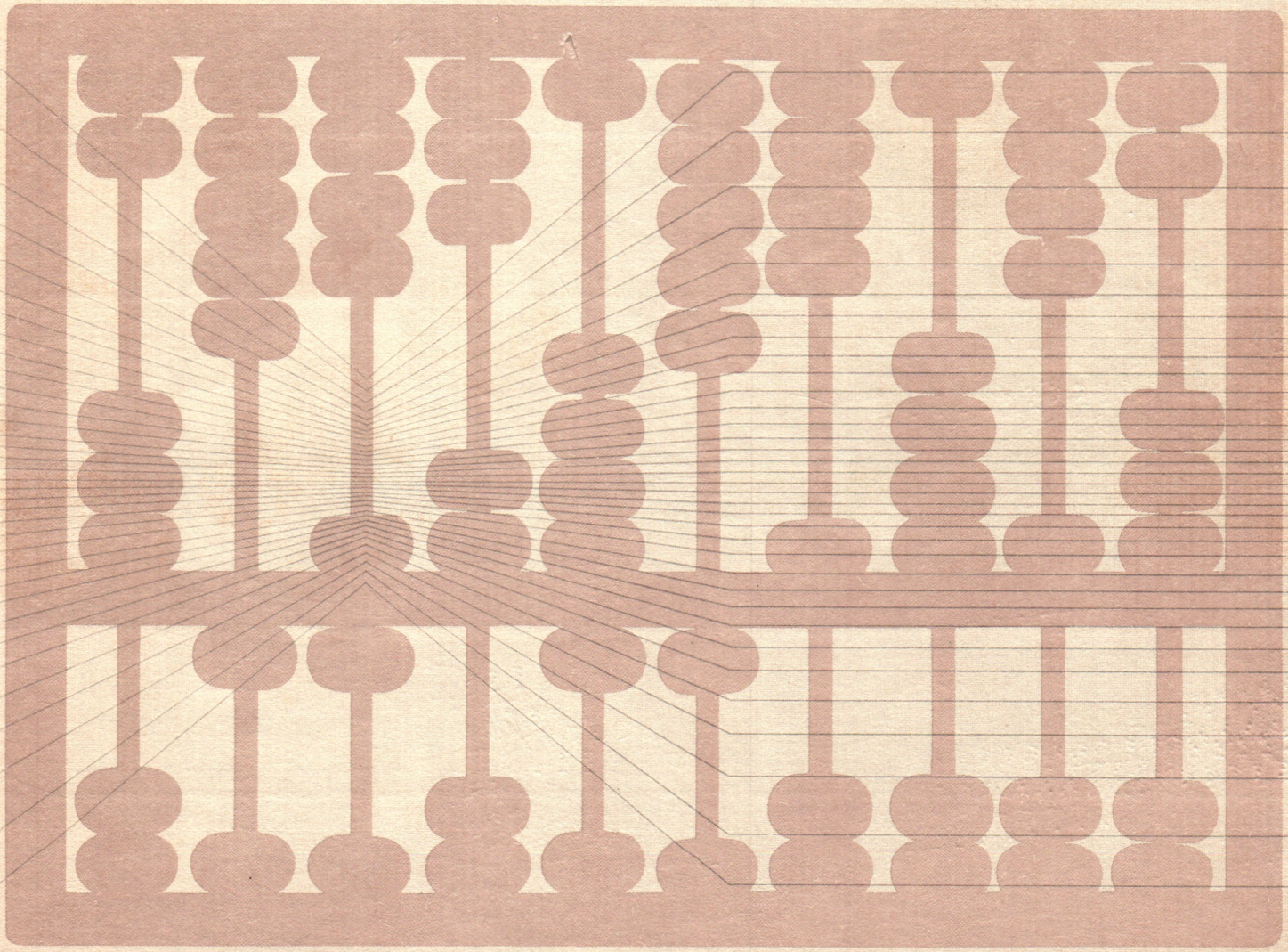
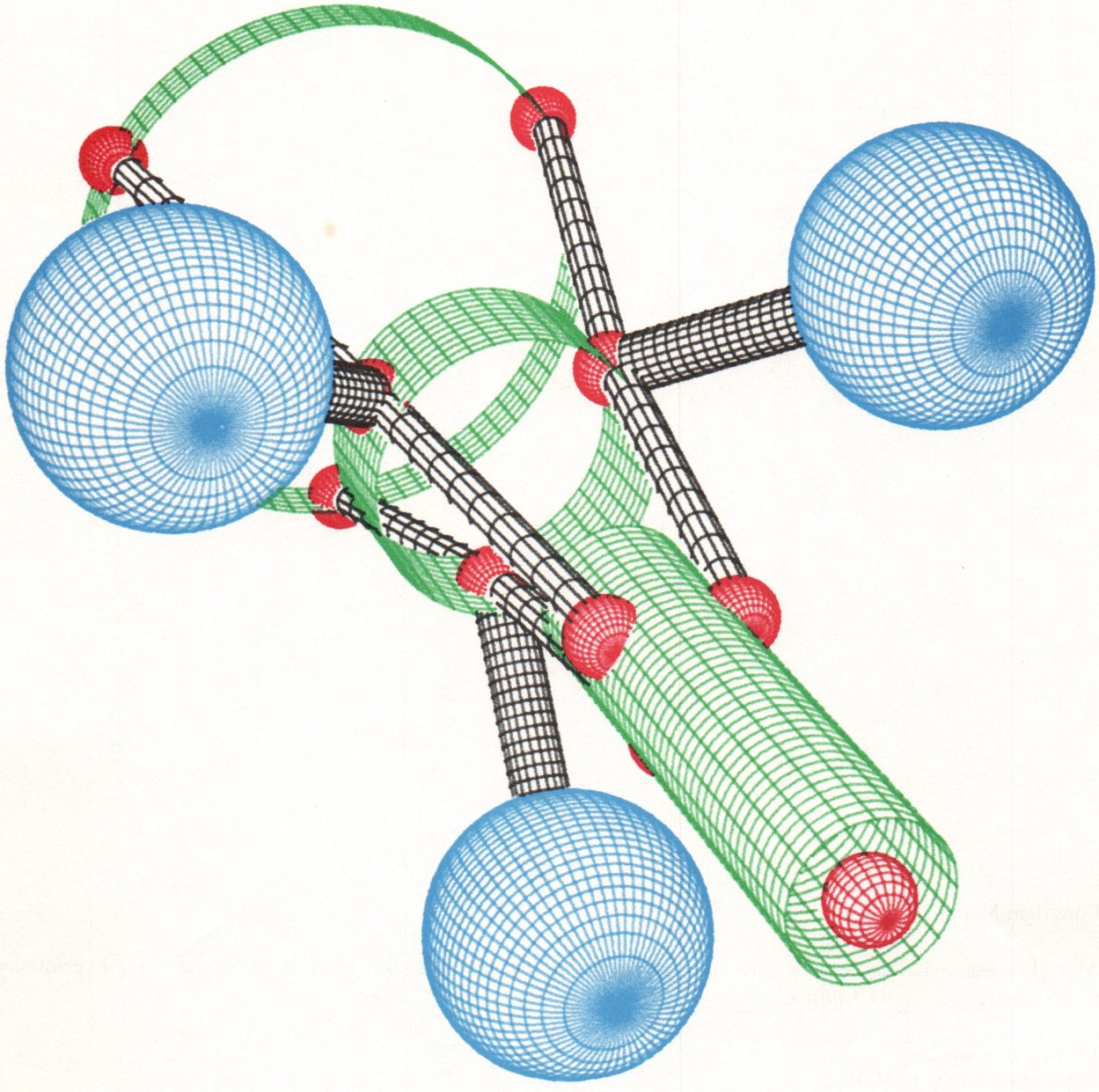
I. P. Sharp Associates

SHARP APL Graphics



SHARP APL GRAPHICS



Copyright March 1978

All rights reserved. This book or parts thereof may not be reproduced by any means without the written permission of I.P. Sharp Associates Limited.

Printed in Canada.
Publication Code: 631GR12

CONTENTS

SECTION I - Introduction to SHARP APL Graphics

I-A	Starting out		3
I-B	At the terminal	- Session 1	23
I-C		- Session 2	43

SECTION II - Textbook

	Key to Abbreviations	47
II-A	Input-Output functions	48
II-B	Transforms	77
II-C	APL-like functions	91
II-D	Utility functions	101
II-E	Three space functions	107
II-F	File functions	115

APPENDIX A - Glossary

Abbreviations used in text	117
Boxes	117
Global variables	118
Objects and graphic vectors	119
Objects on file	120
Text	120
Terminals supported	121
Transforms	122

APPENDIX B - Reference Card

Input-Output Functions and Object Generators	125
Transforms	128
APL-like functions	130
Utility functions	131
Three space functions	132
File functions	133
Global Variables	133

Bibliography 2

INTRODUCTION

SHARP APL forms a good general base for an interactive graphics system. It handles complex parallel computation and provides a simple, efficient file system. The highly interactive nature of APL allows rapid development of new user systems. Feedback is immediate, in contrast to the hours or days typical of a traditional batch/line-printer/plotter approach.

The SHARP APL graphics package is a tool for two-dimensional and three-dimensional line drawing on a wide variety of CRT, hard-copy and plotter terminals.

The functions fall into the following categories:

1) Terminal Drivers, the terminal-dependent functions that perform graphic input and output.

2) Transform, object, and string generators: these generate the graphic data structures used by graphics.

3) Transform applicators: functions that transform pictures by such means as scaling, translation, rotation, clipping, changing pens or type font, boxing, and so on.

4) APL-like functions: these functions manipulate graphic data structures in a manner analogous to APL.

5) Utility functions: functions commonly used in graphics but falling into none of the other groups.

SHARP APL graphics differs from most other APL-based graphics systems in that it allows an object (picture), text string, or transform, to be treated in a manner similar to an APL scalar. This allows creation and manipulation of entire collections of objects at once.

The reader is expected to be well versed in APL, and should have a terminal handy to try the examples as they occur in the text.

Workspace 3 *GRAPHICS* contains the documented versions of the graphics functions. Workspace 3 *GRAFIX* contains compressed versions (without comments) of the same functions.

Note that both workspaces are origin 1 dependent.

ACKNOWLEDGEMENT

The package was written by Bob Bernecky, Peter Wooster and David King of I.P. Sharp Associates in Toronto. Roger Moore contributed heavily to the design in the earlier stages.

Bibliography

Newman and Sproull (McGraw-Hill, 1973) - "Principles of Interactive Computer Graphics"

The Carleton University (Ottawa, Canada, 1976) - "APLGRAF Reference Manual"

SECTION I

STARTING OUT

In order to introduce the user to some of the capabilities (and fun) of the SHARP APL graphics system quickly, a selection of the available functions is discussed in this section in the following sequence:

Making an object,
Drawing the object,
Manipulating the object,
Decorating the object.

SECTION II contains detailed explanations of all functions.

The object of any graphics system is to produce a picture that can be drawn on a terminal screen or paper, seen from different angles, added to, clipped to fit the screen or any other boundaries, and moved about.

Before we start, we have to define the space in which our object rests and the size of the object in terms of its **universe**

UNIVERSE, VIEW, WINDOW

In graphics the **universe** is a Cartesian N -space, typically 2-space (two-dimensional) or 3-space (three-dimensional). Points and line segments can be drawn in this empty space. Any set of points and/or line segments in space can be logically grouped together to form an **object**. SHARP APL graphics can be used to construct and manipulate objects in the universe.

The user views the universe through a **window** bounded by the edges of a physical or logical screen. The global transform TRANS controls the position of the window in relation to the user's space. TRANS maps the user space into a square window of radius 1 with its origin in the centre of the window.
The default value of TRANS is:

```
TRANS←SCALE÷10
```

which results in a user space oriented so that the origin (X,Y=0,0) is at the centre of a window with a radius of 10 units. The **view** through the window is clipped to match the aspect ratio of a rectangular screen (half of the longer side is 10 units).

The global variable CTRANS may be used to divide the physical screen logically into several frames.

The global variable *SMTRANS* describes the terminal and should not be altered by the naive user.

BEGINNER'S VOCABULARY

These abbreviated definitions touch on some of the more important concepts of the SHARP APL graphics package. APPENDIX A (GLOSSARY page 117) expands on these definitions and SECTION II (page 47) contains detailed text.

Object

An object is a collection of point and/or line segments in space. An object is treated as an entity by the graphics system, in much the same manner as a scalar is treated by APL. An object may be thought of as a graphic scalar or a one-element vector. The APL representation of a graphic object, however, is a matrix. The matrix consists of controls and the cartesian coordinates of the points that define the object. Each row defines a point. Column 1 is reserved for object delimiters and pen controls. Columns 2 and 3 specify the X and Y coordinates of the object. For three-dimensional objects a fourth column is needed to specify the Z-axis coordinates. Higher dimensional objects may be created and most transform applicators will properly manipulate them.

Graphic Vector

Zero or more objects, catenated (joined together) along the first coordinate, form a picture or graphic vector. Most graphic functions treat a picture made up of N objects as an N-element vector.

Text

Text is represented in graphics as a vector of character strings. A set of APL-like functions manipulate character string vectors, as they do graphic vectors. The more commonly used APL characters are supported both as hardware (e.g. terminal-generated) and as stroke (drawn) characters. Hardware characters will vary depending on the hardware used. Stroke characters look better and are more expensive than hardware characters.

Box

A box may be created to contain a graphic object or text. Boxes provide an easy and convenient way to move pictures around, or place text. A box may be used as a "window" or "viewport" just as the screen boundaries of a TV monitor act when the camera zooms in on an object.

Transform

There are two ways in which graphic vectors, once created, can be manipulated. The data structure itself can be changed using APL-like functions such as *TAKE*, *DROP*, and *COMP* (compress) to add, remove, substitute and combine objects. Alternatively, the picture described by the graphic vector can be scaled, rotated, or translated in the space in which it is defined by the use of **transforms**. Combinations of scaling, rotational and translational transforms are also possible. Transforms may be described in terms of a camera: scale (zoom), translate (reposition or move) and rotate (tilt or pan).

THE TERMINAL

Most graphic input functions are terminal dependent. Input functions are provided for Tektronix 4000 and Hewlett/Packard 2648A series crosshair input "Gin mode", and the Tektronix 4953 graphic tablet. Most non-CRT terminals do not support graphic input at all.

The following terminal types are supported (today). To condition the workspace, type: *TERM* **'code'**, where **code** is one of the codes given below. For example, when using any Tektronix 4000 series terminal, the first commands after signing on would be:

```
)LOAD 3 GRAPHICS
TERM 'TEK'
```

Terminal	**Code**
Any Tektronix 4000 series terminal	*TEK*
Tektronix 4013	*TEK*4013
Tektronix4015	*TEK*4015
Anderson-Jacobson AJ832	*AJ*832
Xerox Diablo model 1550	*DIABLO*
Xerox Diablo 1620	*DIABLO*
Xerox Diablo 1700	*NEWD*1700
DTC300	*DTC*300
DTC302	*DTC*302
Trendata 4000	*TREN*4000
Trendata 4000A	*TREN*4000*A*
Multiwriter II	*ASL*2
Multiwriter III	*ASL*3
Tektronix 4662 Plotter	*T*4662
Houston Instruments Plotter	*HIPPO*
Hewlett/Packard 7221A	*HP*7221
Hewlett/Packard 9221A	*HP*9221
Spintronic	*SPIN*2
Mitek	*MITEKV*3

To obtain an updated list of supported terminals and their codes, type *TERM* **''**.

Terminal-specific strapping and setup requirements are documented in the *OUTTERM* function, created by successful execution of the *TERM* function.

Supporting new terminal types is rather easy, as there is only one terminal-dependent output function (called *OUTTERM*) in the workspace. Thus, one can run the same set of functions on a different terminal by executing the *TERM* function to define the *OUTTERM* function necessary for the terminal in use.

The illustrations in this manual were produced on a Tektronix 4015 with a 4631 hardcopy attachment.

GRAPHIC INPUT

Objects can be generated in many ways: by hand using APL primitives, using object generator functions, by means of interactive crosshair input or the graphics tablet, and from user functions.

Zero or more objects catenated along the first coordinate form a picture (graphic vector). You will recall that graphic vectors are matrices, with each row defining a point of the object, and the columns defined as follows:

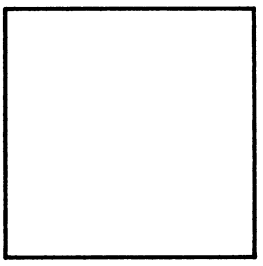
Column 1	Controls	(¯4 for object delimiter,1 for draw, 0 for move - see APPENDIX A on page 119 for other options)
Column 2	X coordinate	
Column 3	Y coordinate	
Column N	*N*-1'th coordinate.	

1) Generating an Object By Hand.

EXAMPLE:

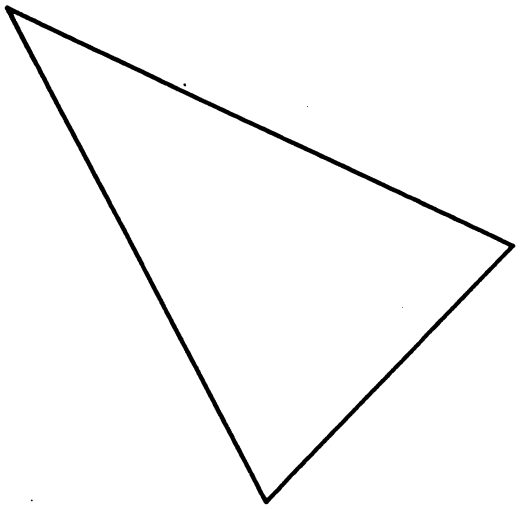
(a) Generate a unit square with the default unit being 1/20th screen width (the screen radius is 10). Notice that the coordinates of the first and last points are equal, to produce a closed figure.

```
      OBJ1←5 3⍴¯4 0 0   1 1 0   1 1 1   1 0 1   1 0 0

      OBJ1
¯4 0 0
 1 1 0
 1 1 1
 1 0 1
 1 0 0

      DRAW OBJ1
```



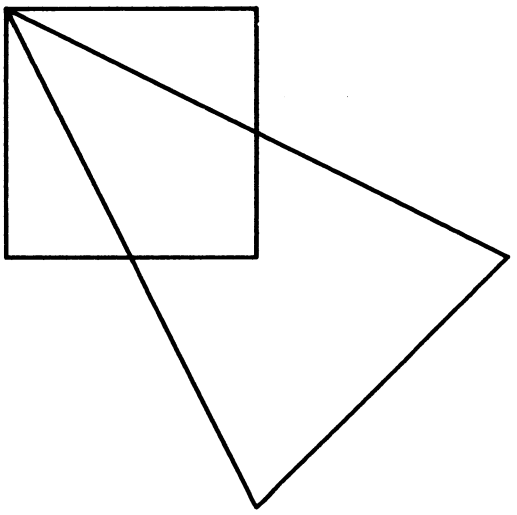
(b) A triangle in two dimensions.

```
      OBJ2←4 3⍴¯4 0 0   1 2 1   1 1 2   1 0 0

      OBJ2
¯4 0 0
 1 2 1
 1 1 2
 1 0 0

      DRAW OBJ2
```



(c) Now create a graphic vector.

```
      GV←OBJ1,[1]OBJ2

      GV
¯4 0 0
 1 1 0
 1 1 1
 1 0 1
 1 0 0
¯4 0 0
 1 2 1
 1 1 2
 1 0 0

      DRAW GV
```



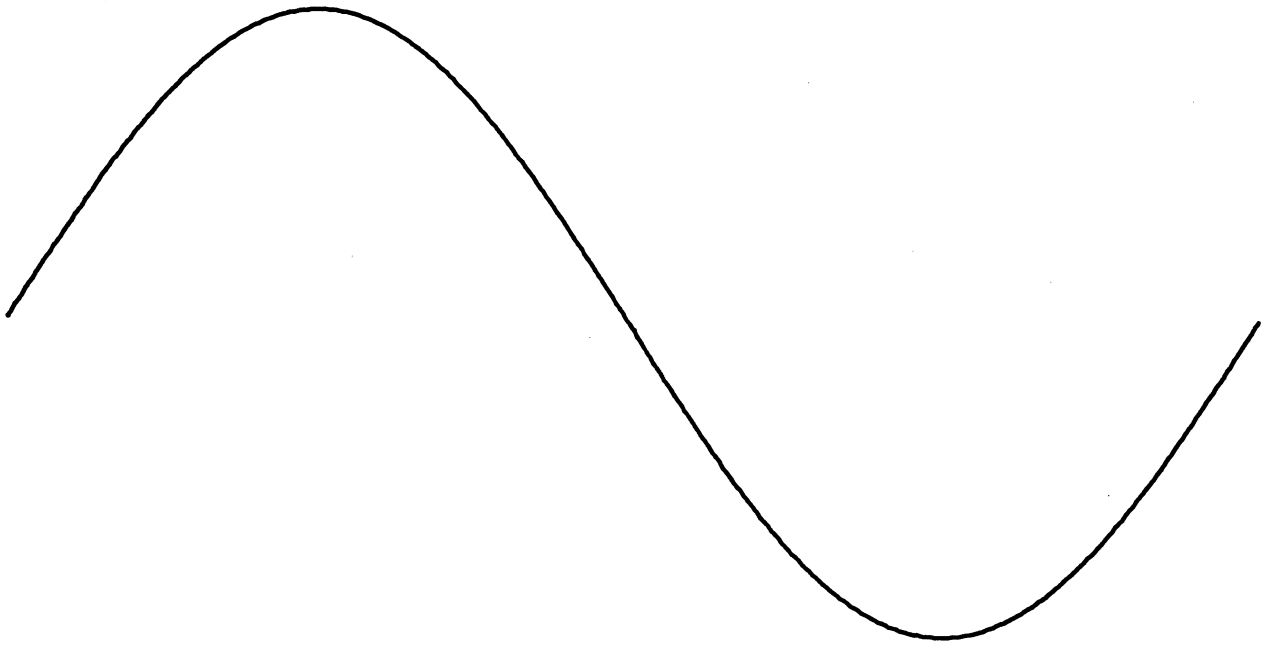
(d) Graphic vectors may be built by the user with simple (or not so simple) APL expressions, for instance the sine curve.

EXAMPLE:

```
GV2←1,((¯181+⍳361)÷18),[1.5]5×10(⍳361)×○÷180
```

NOTE: The global variable *OD* is a vector of controls defining a set of object delimiters. Default is ¯4 ¯2 (object delimiter, point plot object delimiter). Therefore:

```
GV2[1;1]←1⍴OD

DRAW GV2
```



2) Using **object generator functions** provided by SHARP APL graphics. All graphic input functions generate graphic vectors. See the discussion of input-output functions in SECTION II-A (page 48).

(a) The function *MO* (Make Object) generates an object from numeric data.

FUNCTION: **MO**

SYNTAX: *GV←MO XY*

RIGHT PARAMETER *XY*:

- If *XY* is a scalar, vector, or a 1-column matrix, (⍳⍴,*XY*) will be catenated as *X* coordinates, with *XY* becoming *Y* coordinates. Appropriate controls are added to draw all points.

- If 2≤⍴⍴*XY* then *XY* is restructured to be (2↑(⍴*XY*),1 1)⍴*XY*. Appropriate controls are then added to draw all points.

EXAMPLES: *MO* is handy for plotting - see page 48 for more.

```
      □←DATA←.1×10?50
1.3  2.2  0.6  1.0  0.8  3.8  3.5  5  3.7  4.5

      DRAW MO DATA
```

To ensure that the plot fits on the screen, just put the resulting object in a box:

```
      DRAW BOX BX←¯1  ¯1   1  1 ×6          DRAW (MO DATA) INBOX BX
```

(b) Polygons

FUNCTION: **POLY**

SYNTAX: *GV*←*N*1 *POLY N*2

LEFT PARAMETER *N*1: Scalar or vector specifying the number of sides to the polygon(s).

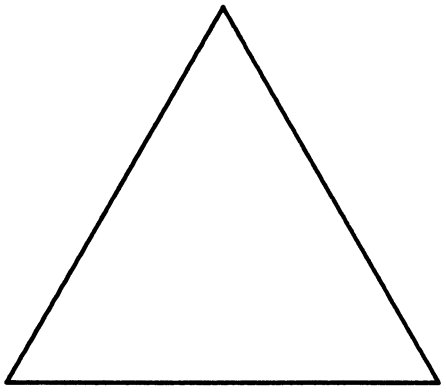
RIGHT PARAMETER *N*2: Radii of the polygons.

NOTES: (*SHAPE GV*)=(⍴*N*1)⌈⍴*N*2
The shorter argument is extended using *RESHAPE*.

EXAMPLES:

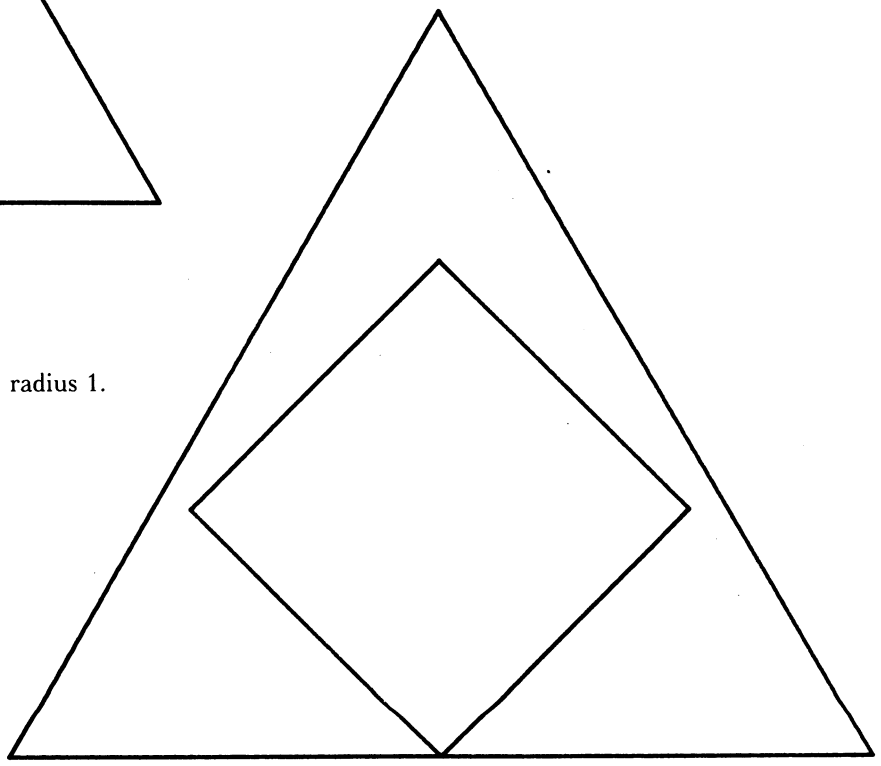
Triangle of radius 1.

```
DRAW 3 POLY 1
```

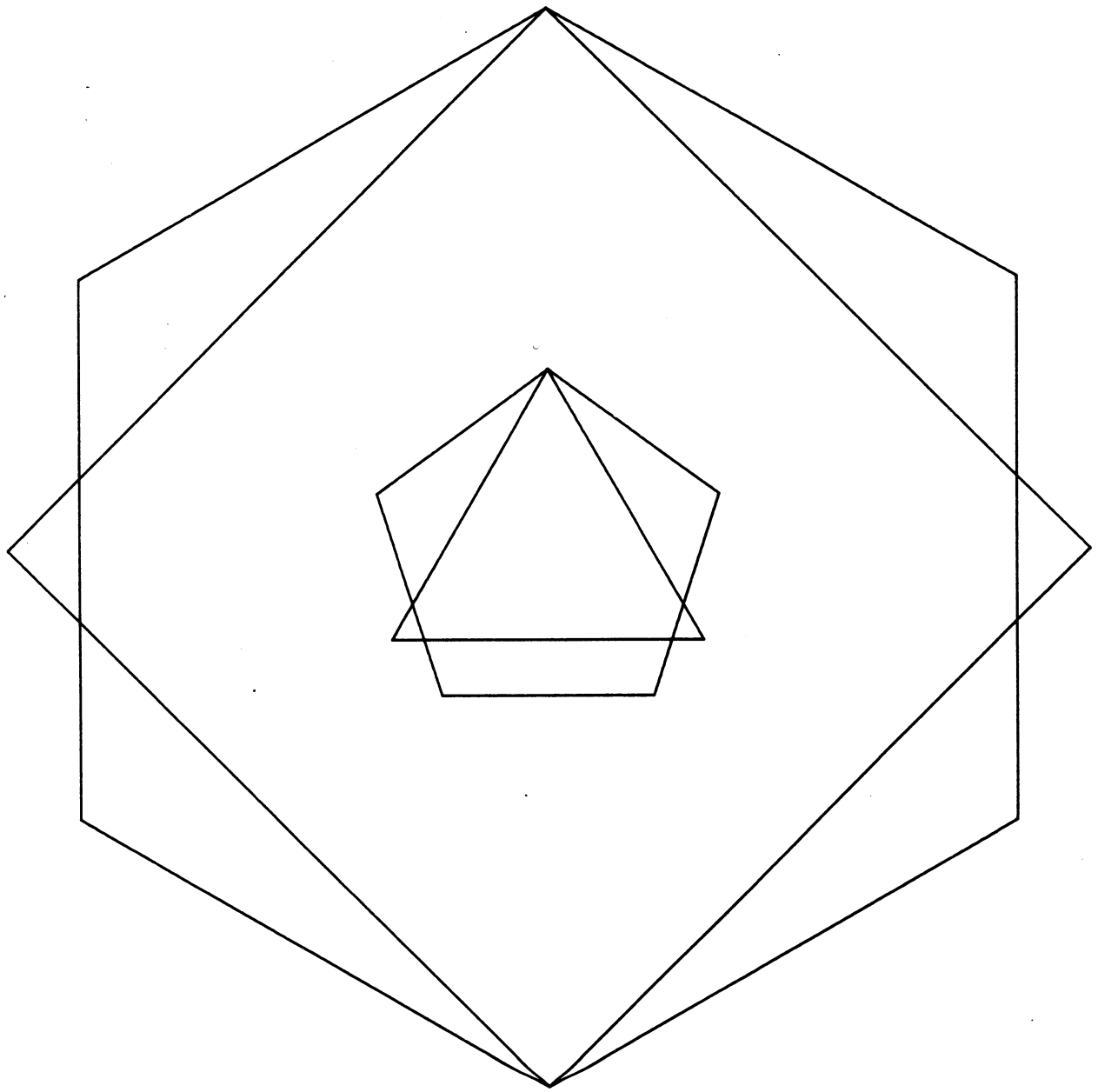


Triangle of radius 2 and diamond radius 1.

```
DRAW 3 4 POLY 2 1
```



DRAW 3 4 5 6 *POLY* 1 3



(c) Daisies and stars

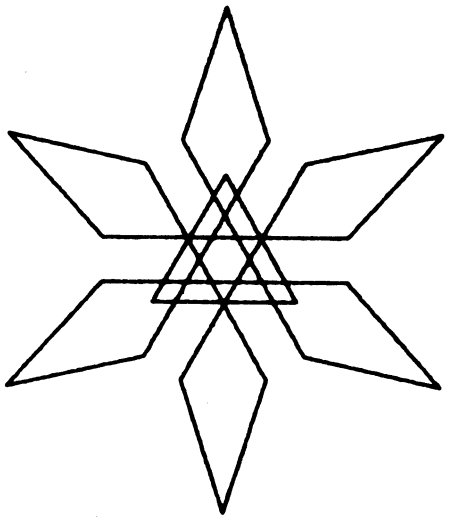
FUNCTION: **DAISY**

SYNTAX: *GV←N1 DAISY N2*

LEFT PARAMETER *N1*: Number of data points per daisy and number of petals per daisy. *N1* may be a scalar, a 2-element vector, a one-column matrix, or a two-column matrix. A scalar argument is extended with *RESHAPE*. See Section II-A (page 50) for details.

```
DRAW (2 2⍴3 3 18 6) DAISY 1 3
```

Points	Petals
3	3
18	6

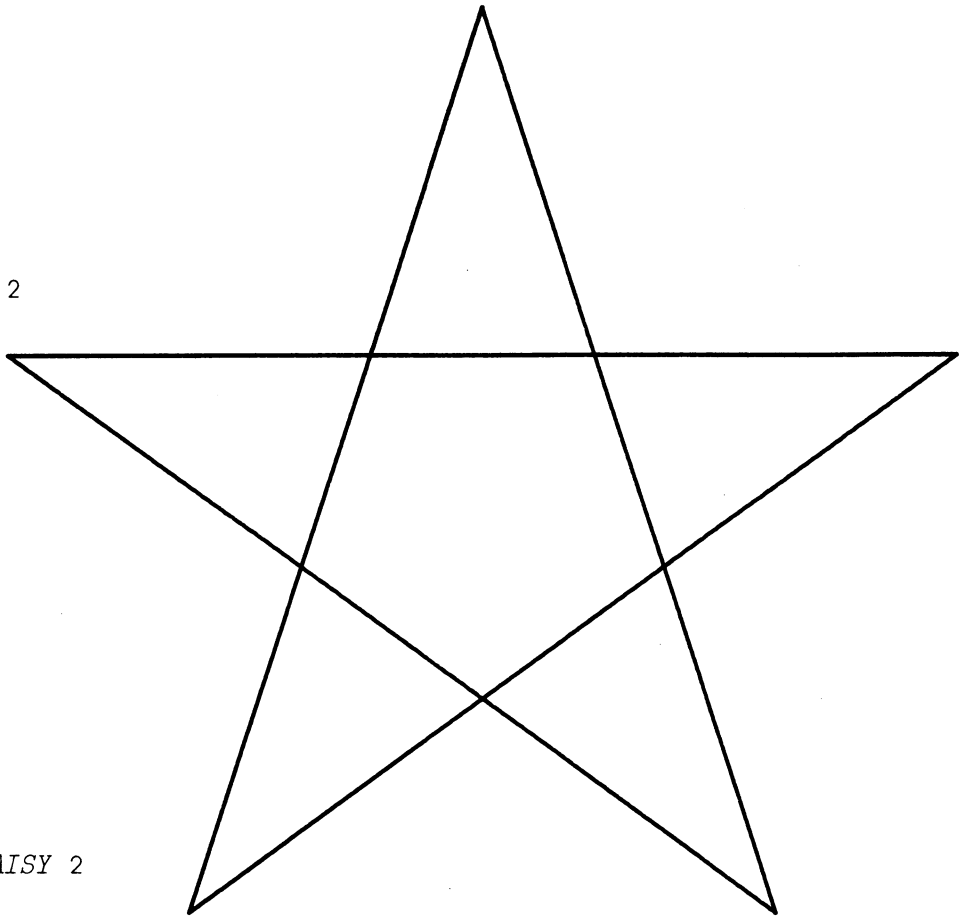


RIGHT PARAMETER *N2*: Radii of daisies. *N2* may be a scalar or a vector. When building more than one daisy, *N2*[*I*] is the size of the I'th daisy.

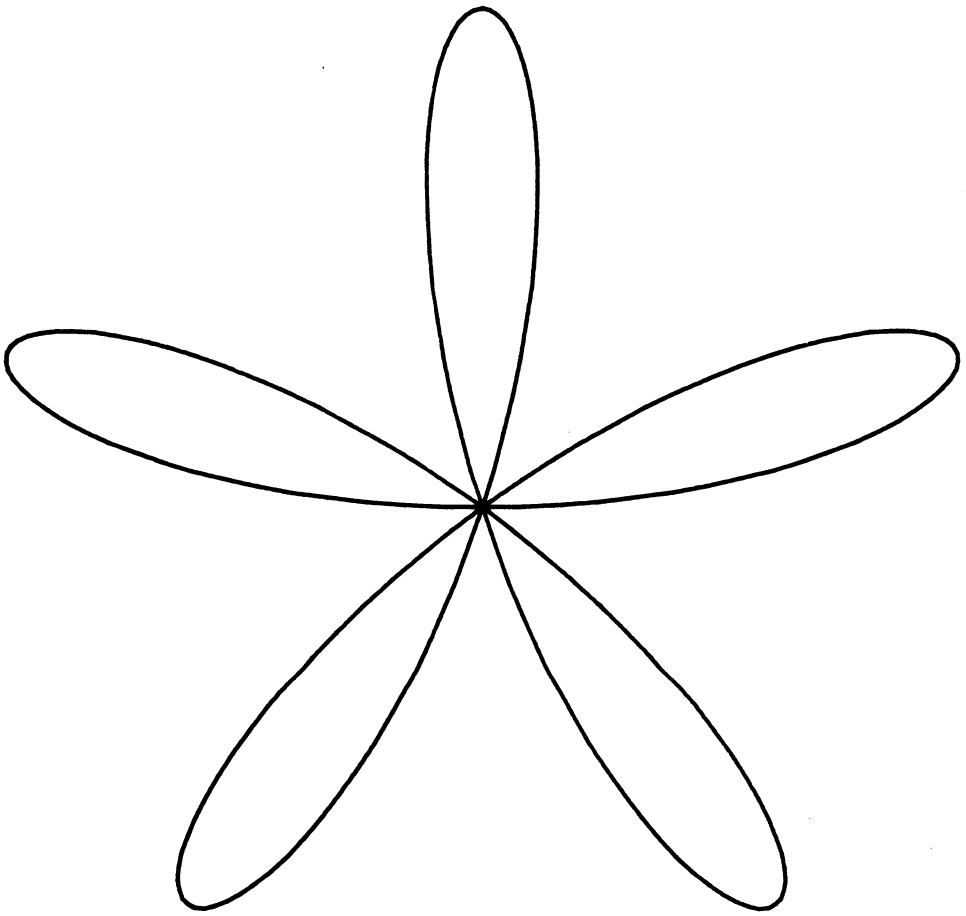
NOTES: (*SHAPE GV*)=(⍴*N2*)⌈1↑⍴*N1*

EXAMPLES:

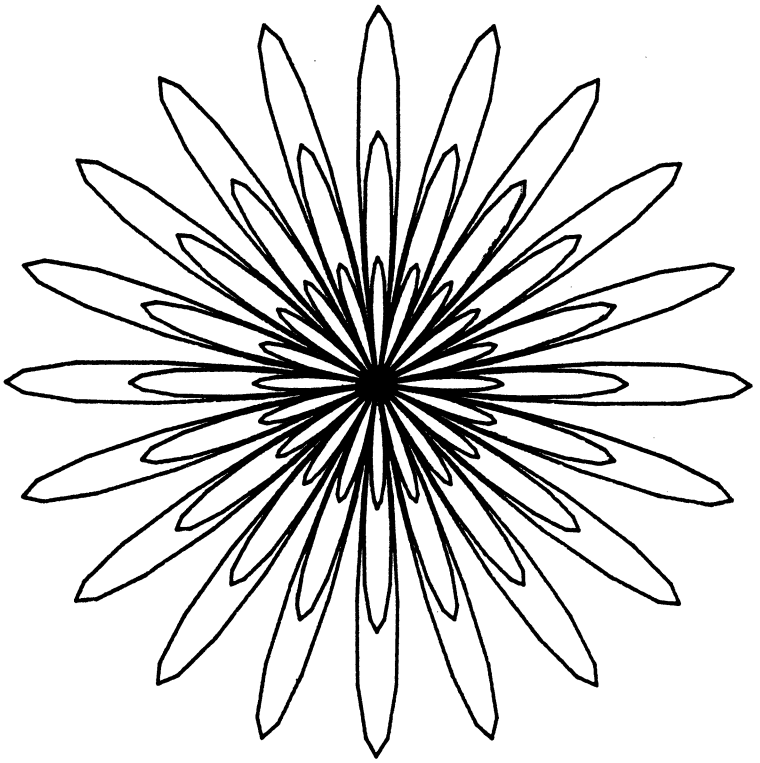
DRAW 5 *DAISY* 2



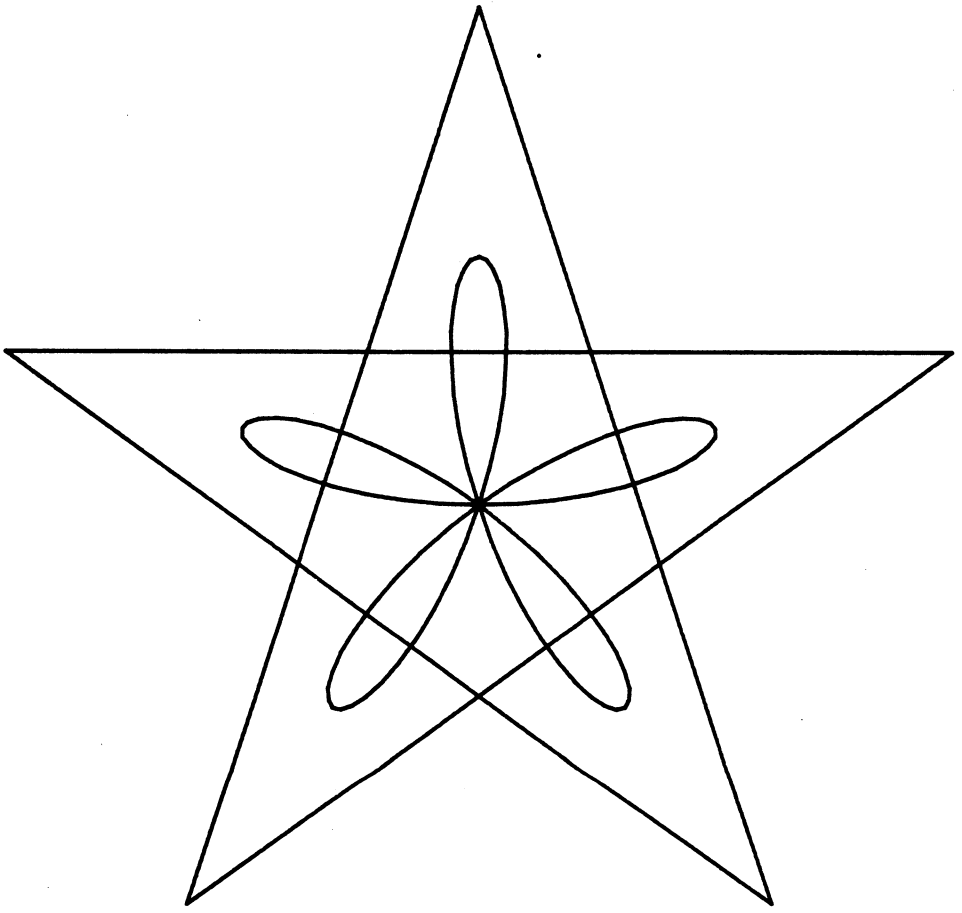
DRAW 100 5 *DAISY* 2



DRAW 100 20 *DAISY* ⍳3



DRAW (2 2⍴100 5 5 5) *DAISY* 1 2



3) **Interactive Input** is supported on the Tektronix 4013/4015 and Hewlett/Packard 2648A.

(a) **Crosshair input** via the function *XIN* allows the user graphic input via crosshairs, and returns a graphic vector in the user's coordinate system. See SECTION II-A (page 52) for a detailed explanation.

FUNCTION:	**XIN**
SYNTAX:	*GV←CH XIN N*
LEFT PARAMETER *CH*:	Acceptable input characters.
RIGHT PARAMETER *N*:	Maximum number of points allowed to complete the picture (0 is no limit).
NOTE:	Blank terminates input prematurely. Carriage return, break, or other ASCII control characters give unpredictable results. Required terminal strapping is documented in the *OUTTERM* function.

EXAMPLES:

`'NPLC' XIN 20`	will create a graphic vector with a maximum of 20 points in the picture. Acceptable input characters are *N*, *P*, *L*, and *C*. The value of an input control generated by an input character is determined by its origin 0 index in *CH*. Thus the controls produced by *N*,*P*,*L* and *C* are 0,1,2 and 3 respectively.

(b) **Crosshair input**: the function *FOLLOW* performs sketch-pad input. It is similar to *XIN* but the resulting graphic vector is drawn as it is generated.

FUNCTION: **FOLLOW**

SYNTAX: *GV←FOLLOW*

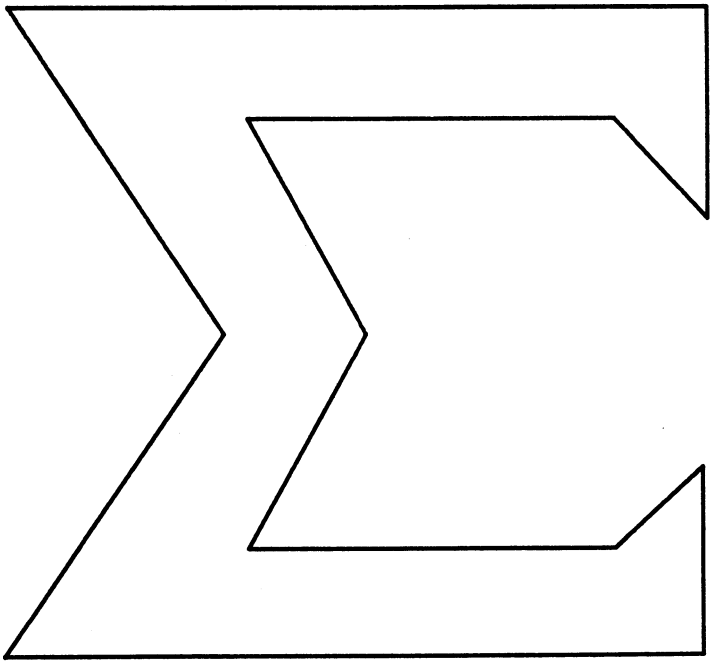
NOTES: Input control characters on the Tektronix keyboard are:
Numerics used directly as controls:

`0,2,4,6,8` = Move to crosshairs
`1,3,5,7,9` = Draw to crosshairs (from previous point).

N Begin new object (i.e. *CONTROL* is `1⍴OD`)
P Draw to nearest point
L Draw to nearest line
C Circular arc through previous, current and next points.

EXAMPLES:

A←FOLLOW Crosshairs appear on the screen. Move the crosshairs and key in control characters as necessary to register points. Typical output using this technique is the drawing below.



SECTION II-A (page 52) contains explanations of the other functions in this group. APPENDIX B (page 125) contains an index of functions.

(c) **Graphics tablet** - pen input on the Tektronix 4953 only. Plugging requirements are documented in the *TABLET* function.

FUNCTION: **TABLET**

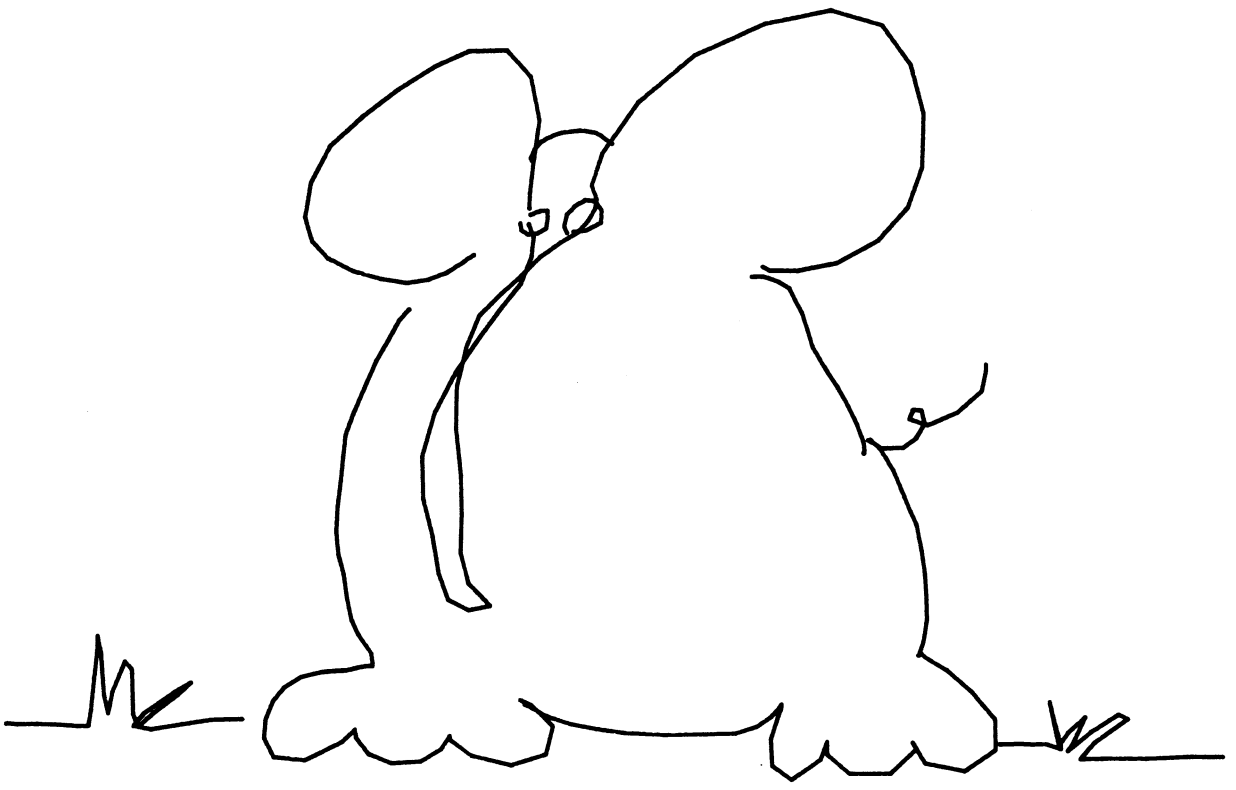
SYNTAX: *GV←TABLET N*

RIGHT PARAMETER *N*:

If *N*=0: Point to point input (connect the dots); each push of the pen inputs **one** point. Removing the pen from the tablet terminates input. The result is a single object, containing as many points as there are pen pushes.

If *N*=1: Continuous input (curve or feature tracing). Points are input at several points per second for as long as the pen is pushed. Each pen push produces an object delimiter, allowing several features to be entered at once. The result contains as many objects as there are pen pushes.

EXAMPLES:



GRAPHIC OUTPUT

1) Drawing Graphic Objects

As seen in the previous examples, DRAW draws graphic vectors (as opposed to text).

FUNCTION: **DRAW**

SYNTAX: *DRAW GV*

RIGHT PARAMETER *GV*: If the graphic vector to be drawn is an Nx3 matrix, it is clipped to the screen boundaries, and drawn.

Drawing from Files

SYNTAX: *DRAW F*

RIGHT PARAMETER *F*: If *F* is a vector, read and draw the components `1↓`*F* from the file tied to `1↑`*F*.

If *F* is a scalar or one-element vector, read and draw all components of the file tied to *F*. Character components and packages are ignored.

2) Drawing Text

FUNCTION: **DRAWSTR**

SYNTAX: *BX DRAWSTR SV*

LEFT PARAMETER *BX*: Zero or more boxes to set the boundaries for the text to be drawn. For example, *BX* may be a rectangle with the bottom left-hand corner at screen coordinates (0,0) and the top right-hand corner at the point (1,5). *BX* would be defined as follows:

```
BX←0 0 1 5
```

RIGHT PARAMETER *SV*: Text string vector. Strings are delimited by the first character in the first string. The first character after each string delimiter is the type font (for example, on all terminals, 0 is APL hardware characters, and 8 and 9 are stroke or drawn characters).
See SECTION II-A (page 56) for more details on *SV*.

Each string in *SV* is drawn such that the first character in the string is located at the upper left-hand corner of the box. Stroke characters are scaled to fit the box.

EXAMPLE:

```
(3 4⍴0 0 5 1) DRAWSTR '/0ONE/8TWO/2THREE'
```

The other functions in the group **Input-Output functions**, (TEXT, VTEXT, STR for text output), are discussed in more detail in SECTION II-A (page 48). See Appendix B (page 125) for a complete list of functions in this group.

MANIPULATING OBJECTS

The essence of interactive graphics lies in the way the user can manipulate each picture, or part of a picture. Objects can be **transformed**, or they may be manipulated via APL-like functions. They may subsequently be decorated, placed on file, flipped, and otherwise abused.

Transforms that will rotate, scale, or translate (reposition) the picture are discussed in detail in SECTION II-B (page 77). The two transforms that appear in the following sessions at the terminal are translation, via the function *IAX*, and rotation via the function *IAR*.

This chapter and the session at the terminal that follows, introduce the capabilities of the system. SECTIONS II-B (page 77) and II-C (page 91) should be read carefully for an understanding of transforms and APL-like manipulations. APPENDIX A (page 117) contains short definitions and APPENDIX B (page 125) lists all functions in a reference card format.

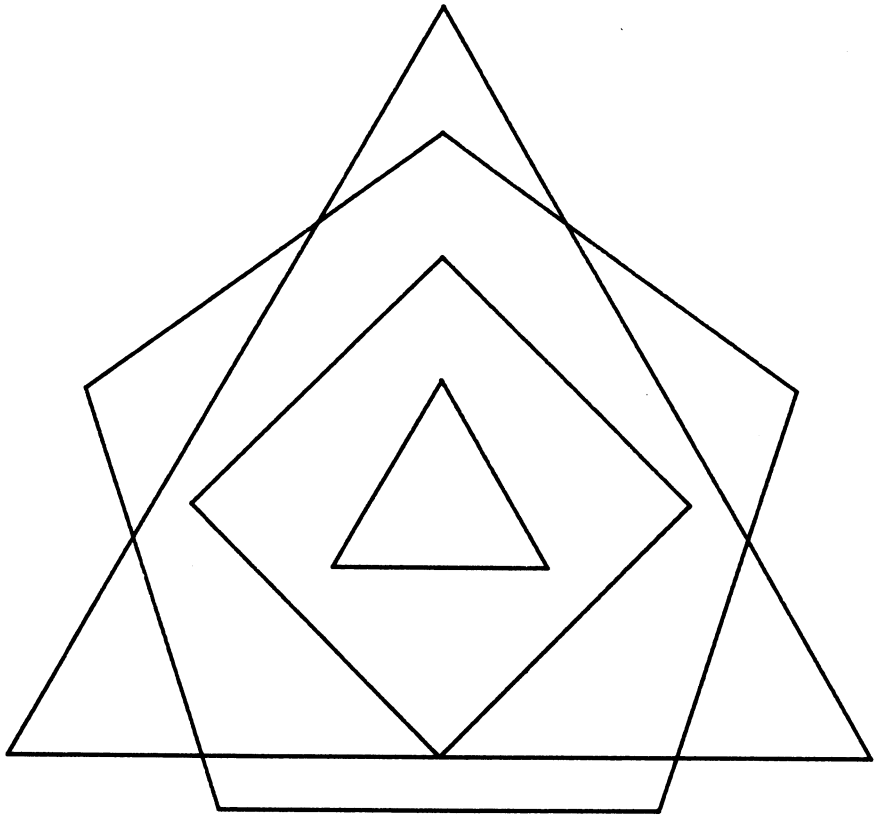
The APL-like functions demonstrated on the following pages are:
TAKE, *DROP*, *COMP* (compress), *MOS* (make object scalar), *PMOS* (partitioned *MOS*), *SHAPE* (monadic ⍴ for graphic vectors), *ISHAPE* (itemwise *SHAPE*), *NDXG* (index generator), and an object decorator from the group *HATCH*. SECTION II-C (page 91) contains discussions of all functions in this group.

The text placement function *TEXT* and object generator *DAISY* are also demonstrated.

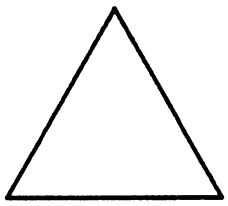
The following sessions at the terminal show some of the capabilities of the SHARP APL graphics package. An object is generated and manipulated first by means of APL-like functions, and then by transforms. The user is encouraged to try the manipulations on a series of objects.

SESSION 1

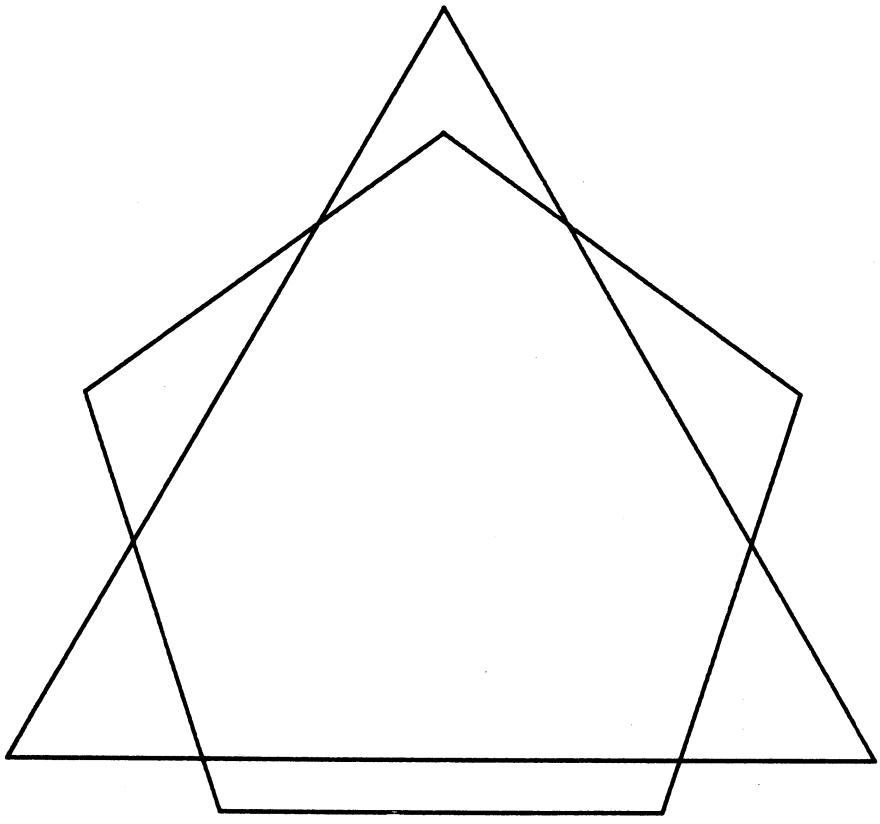
```
      )LOAD 3 GRAFIX
SAVED 00.58.17  18/11/77

      TERM 'TEK'

      A←3 4 5 POLY ⍳4
      DRAW A
```



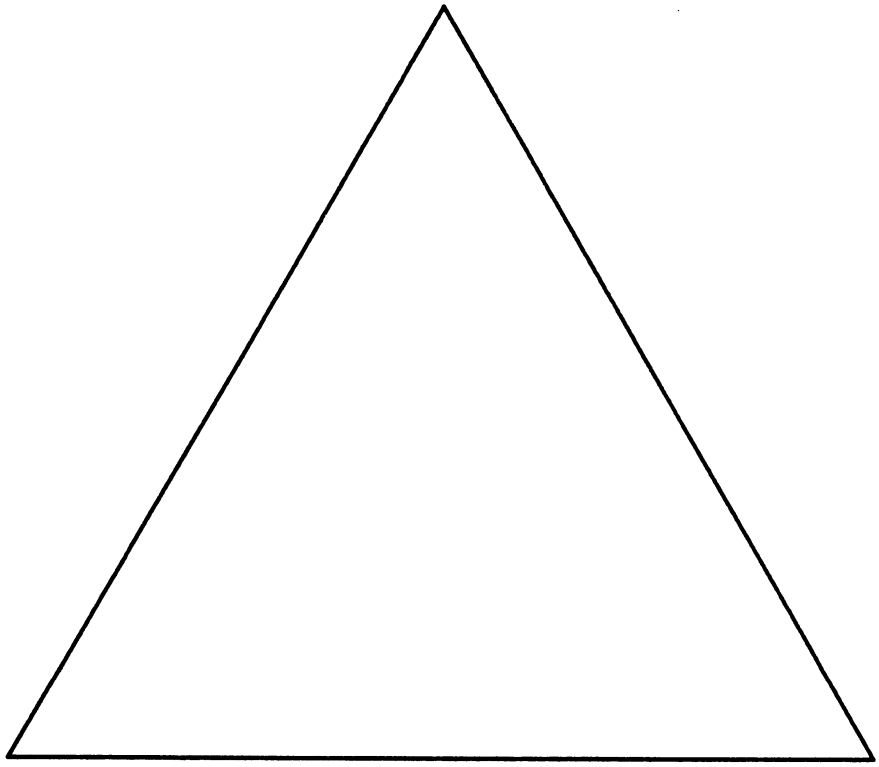
DRAW 1 *TAKE A*



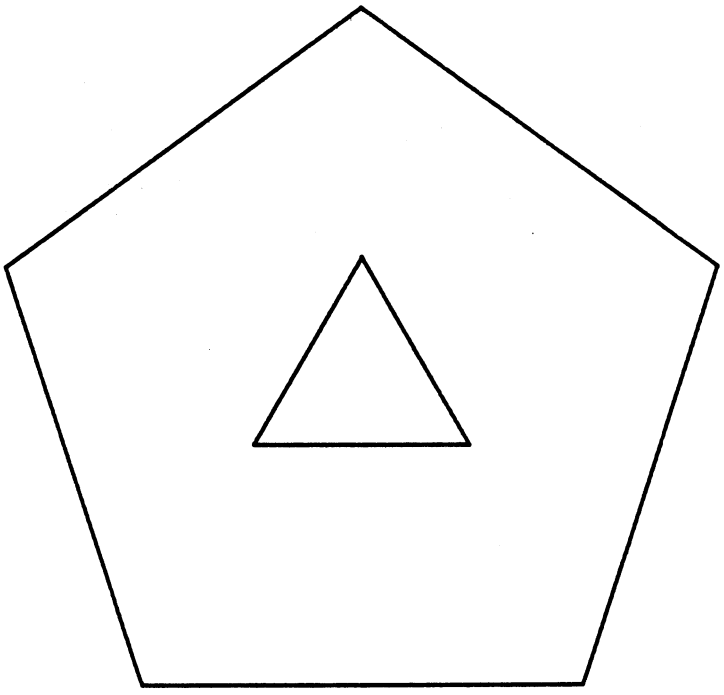
DRAW ¯2 TAKE A



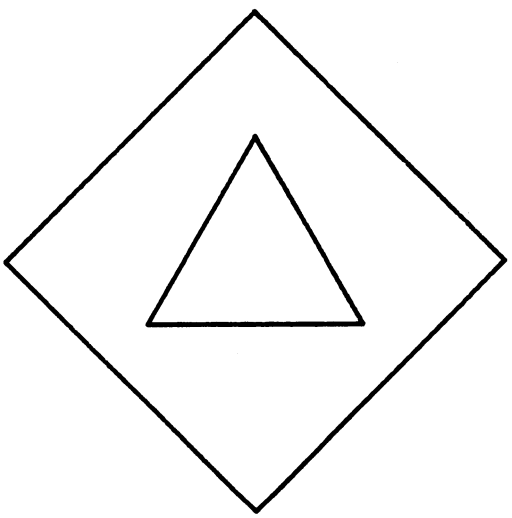
DRAW 3 *DROP A*



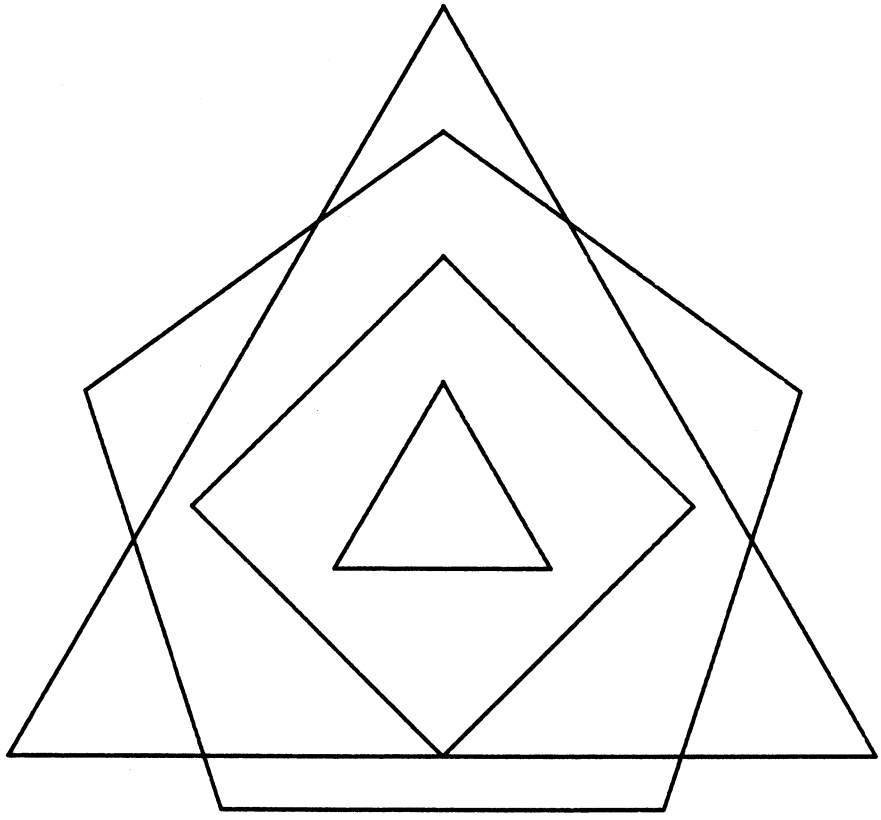
DRAW 1 0 1 0 *COMP A*



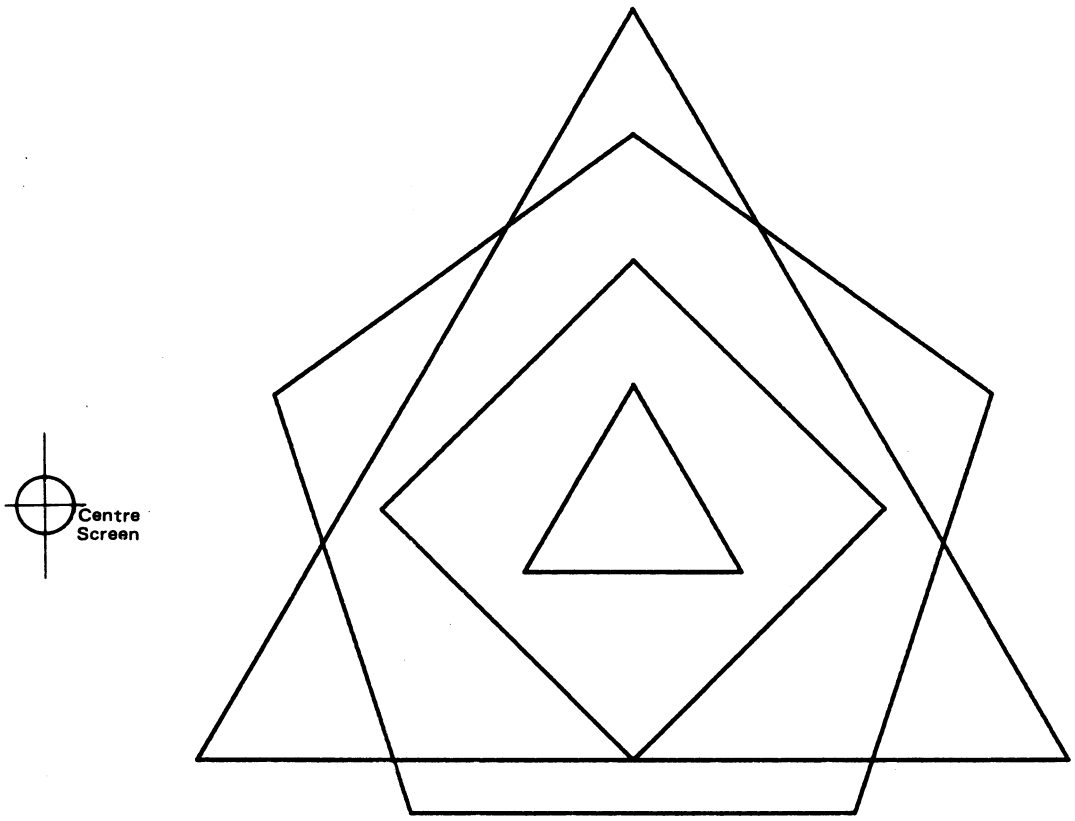
```
DRAW 1 TAKE 1 0 1 0 PMOS A      ACONTROLLED RAVEL
```

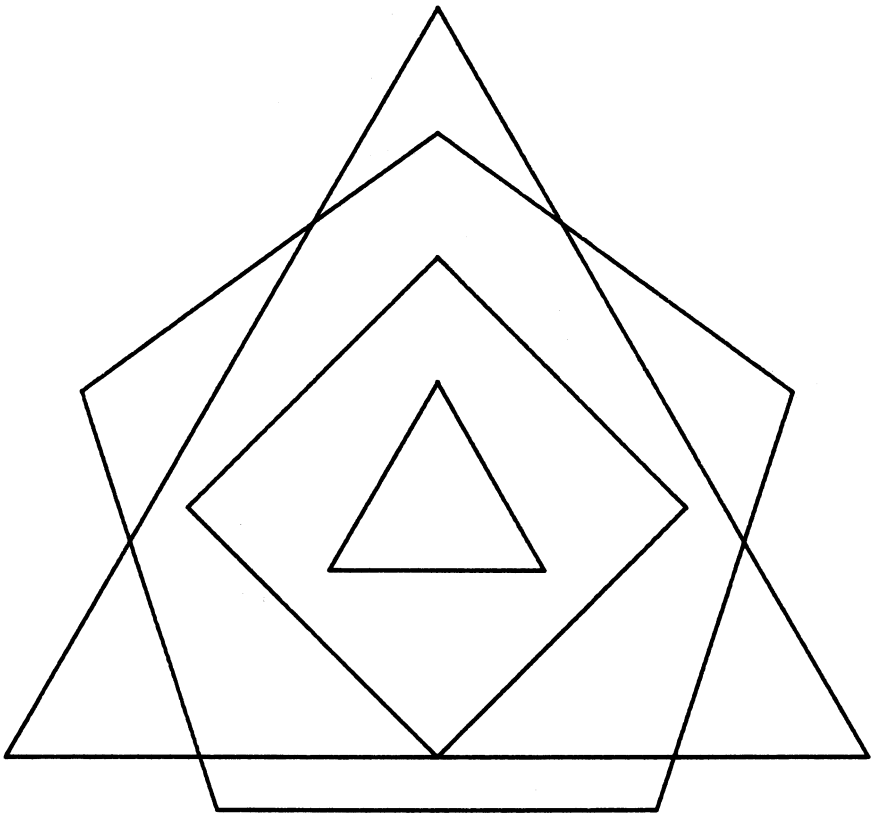


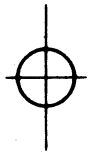
DRAW 1 TAKE MOS A ⍝*RAVEL*



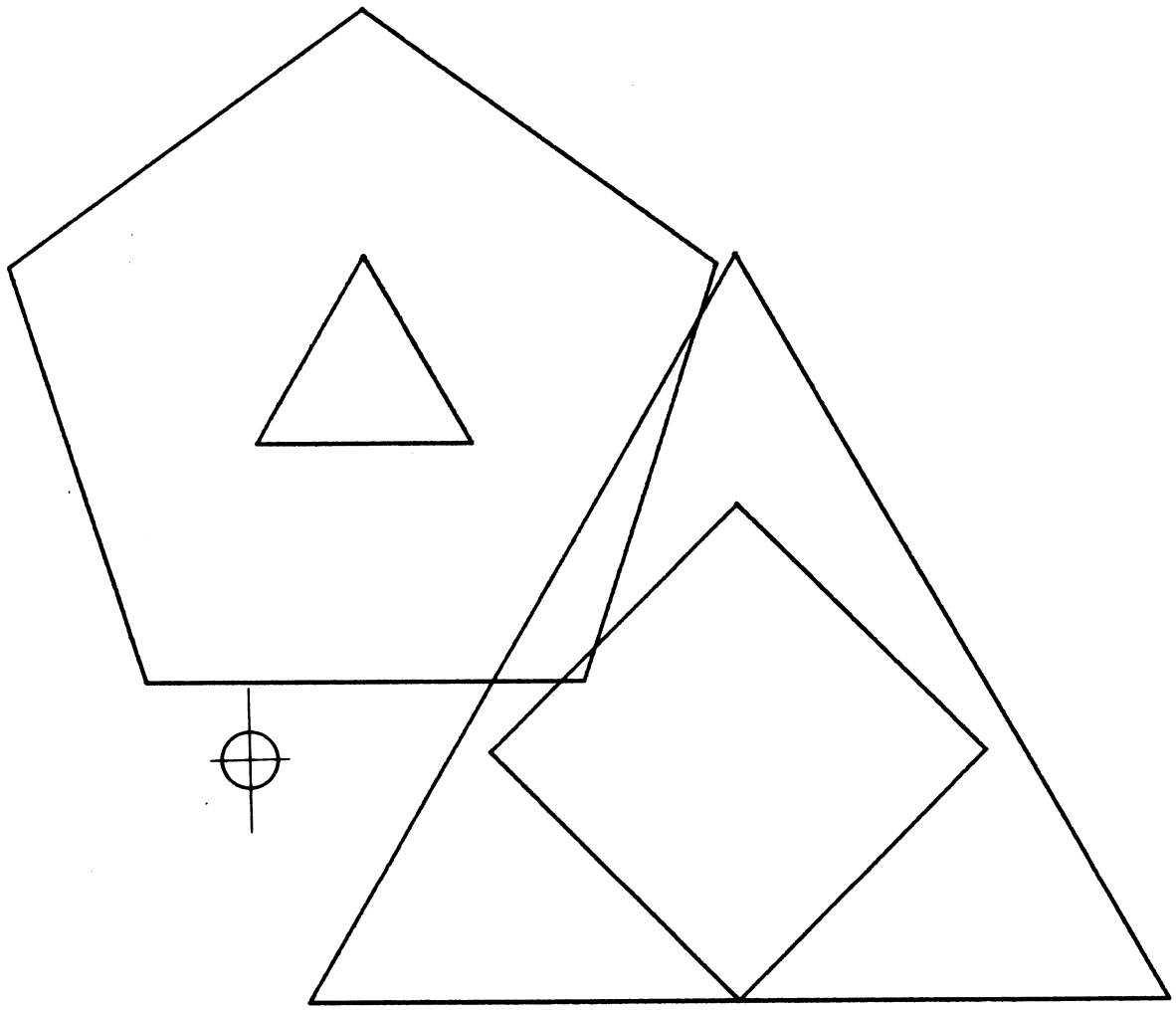
DRAW A IAX 5 ATRANSLATION IN ONE DIMENSION





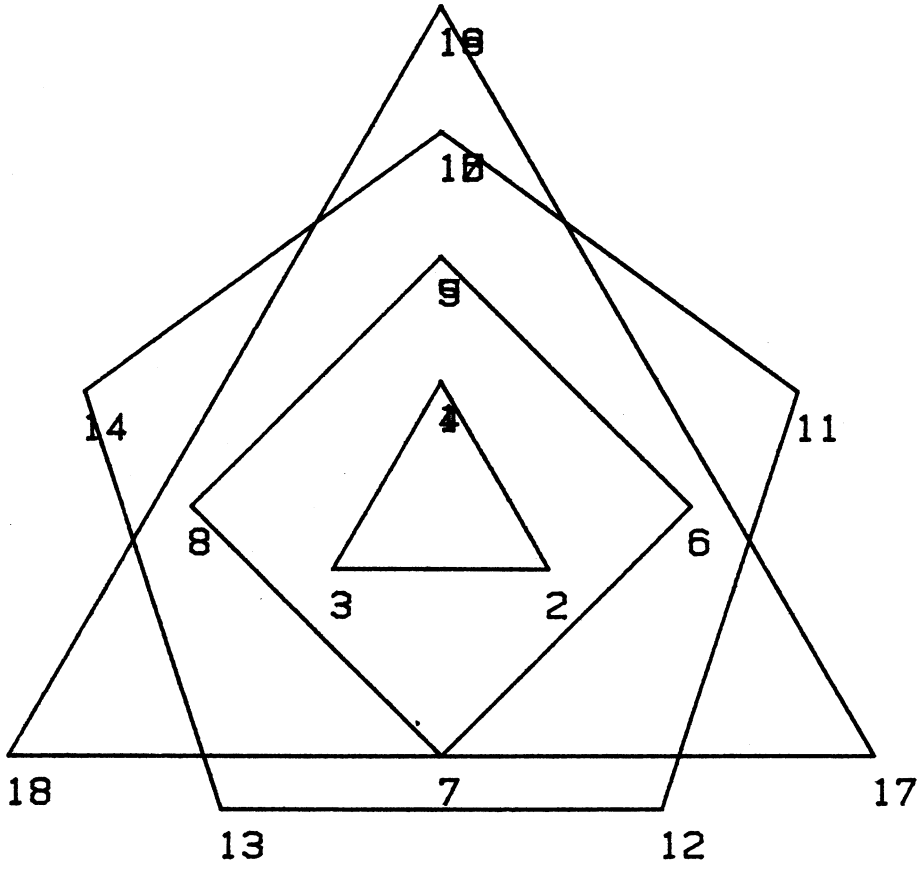


```
DRAW A IAX 2 2 ⍴ 1 3 4 0    ⍝ITEMWISE TRANSLATION BY OBJECT
```



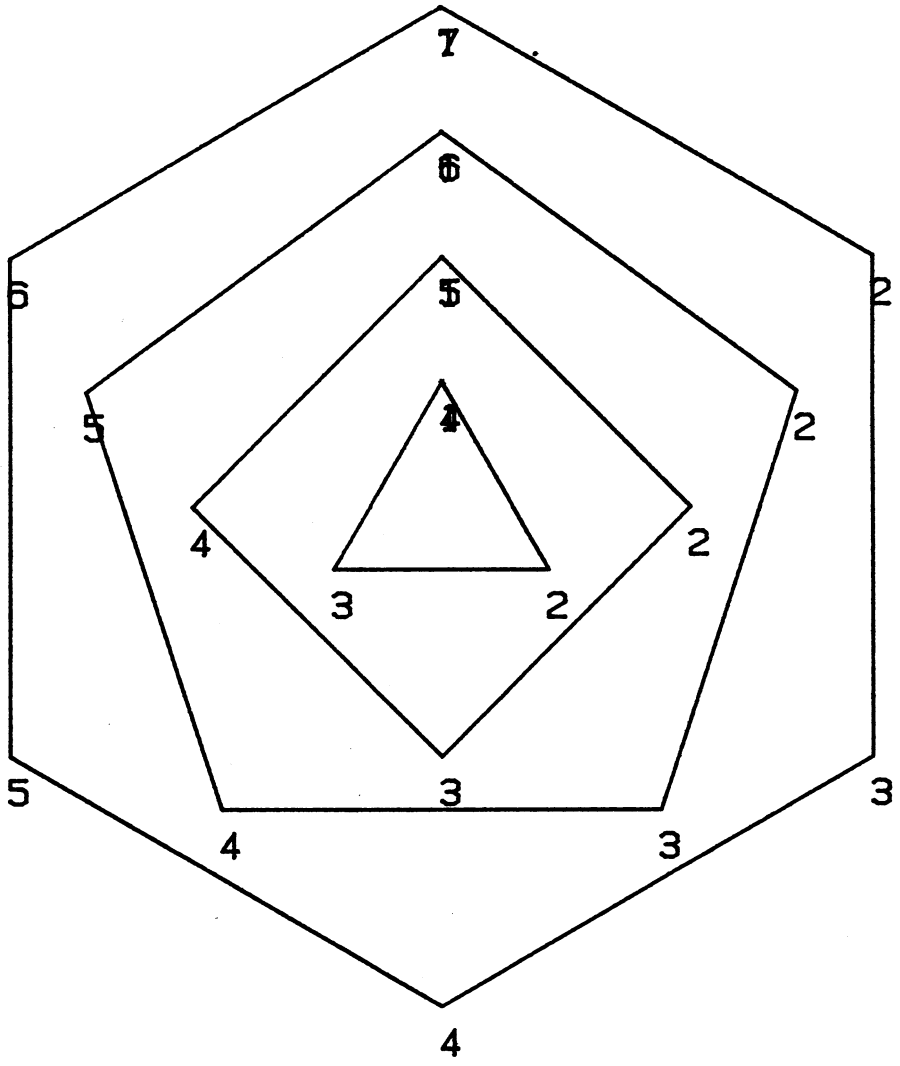
Number each point in A. Dropping the controls from A leaves us with the coordinates of each point.

```
PTS←⍳1↑⍴A
DRAW A ◇ (0 1 ↓A) TEXT ,'</0>,LI4' ⎕FMT PTS
```

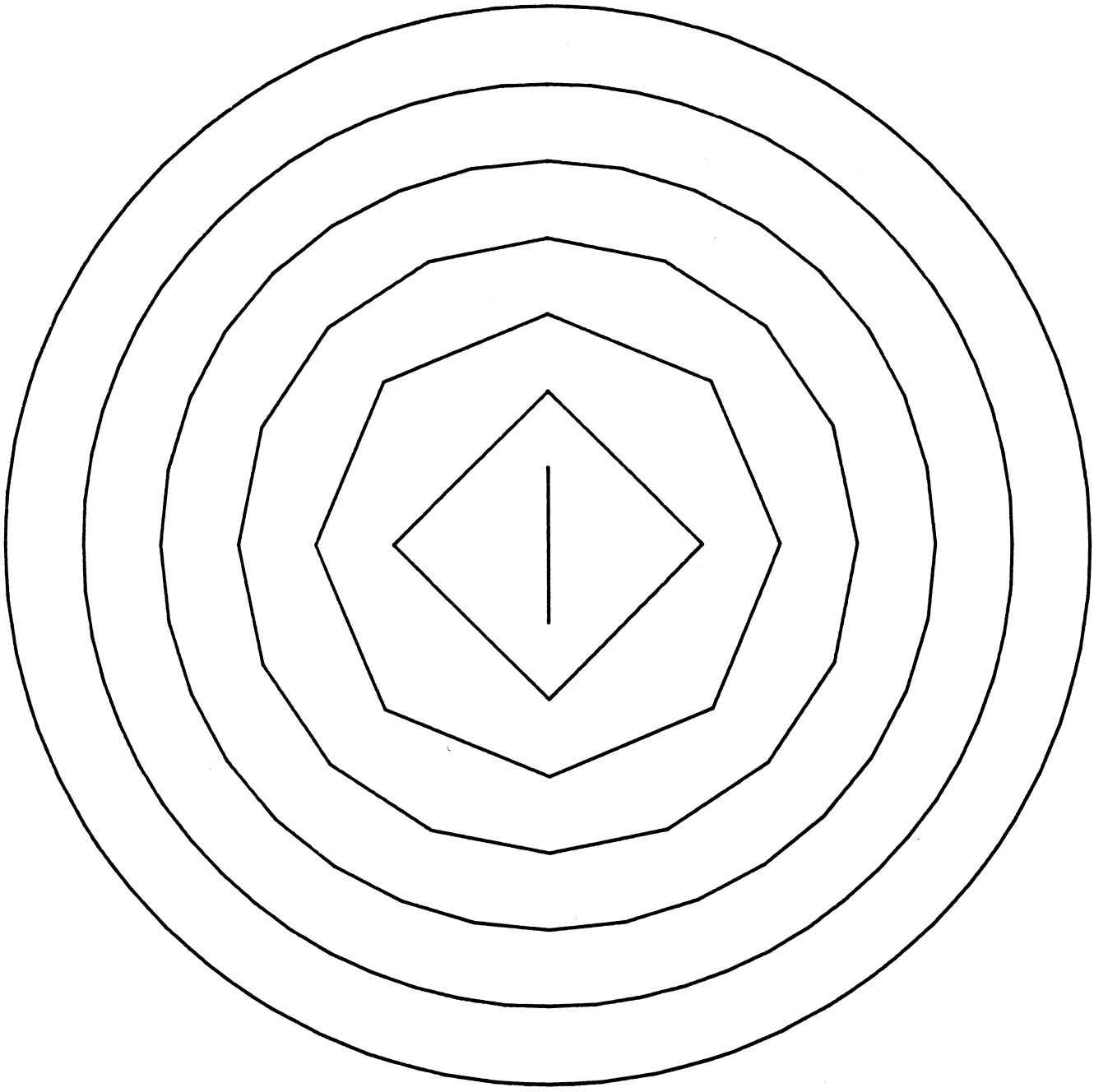


Number each point in A, by object.

```
A←(2+⍳4) POLY ⍳4
PTS←NDXG 1,[1.5] ISHAPE A

DRAW A ◇ (0 1↓A) TEXT ,'</0>,LI4' ⎕FMT PTS
```

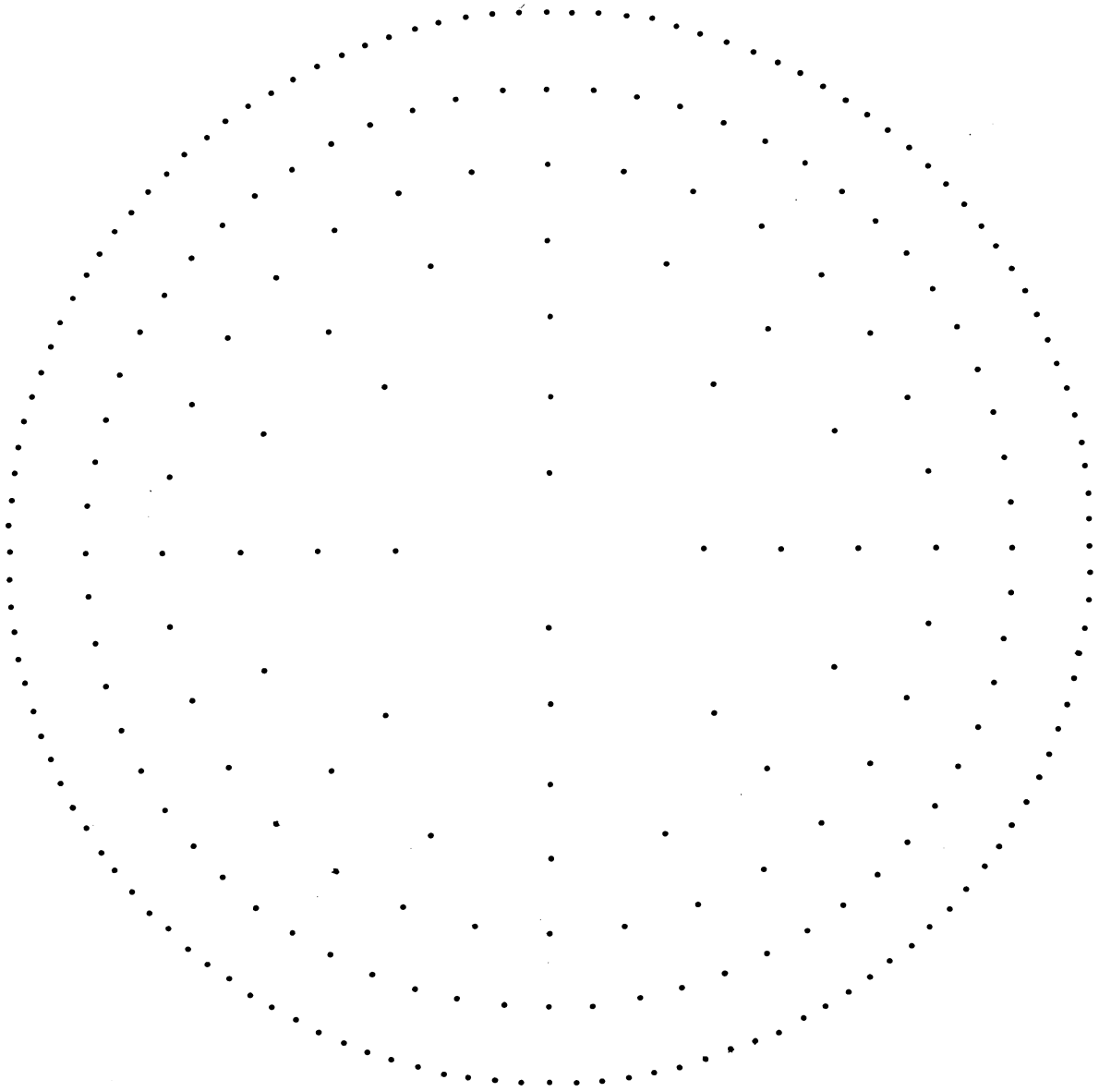


Approximations of circles.



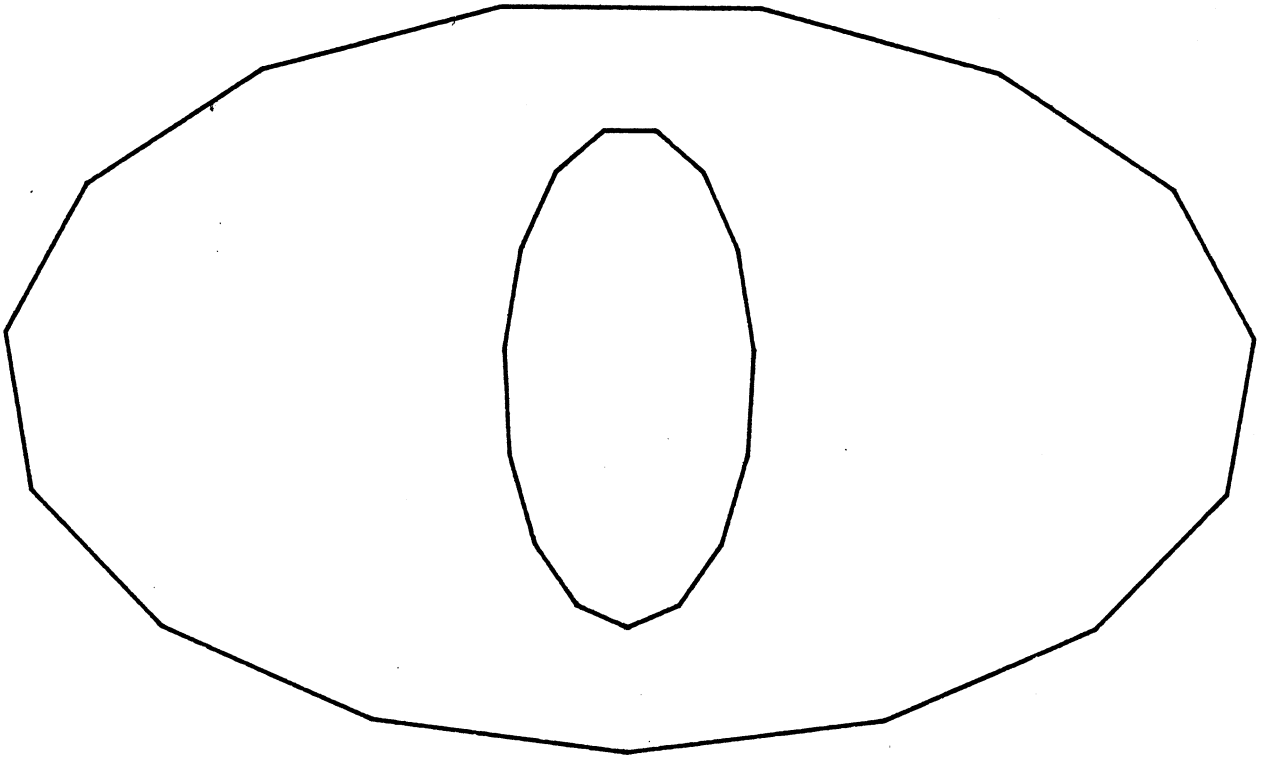
Point plot circle (make each *DRAW* control a point plot control).

```
A[1+¯1↓⍳1↑⍴A;1]←2
DRAW A
```



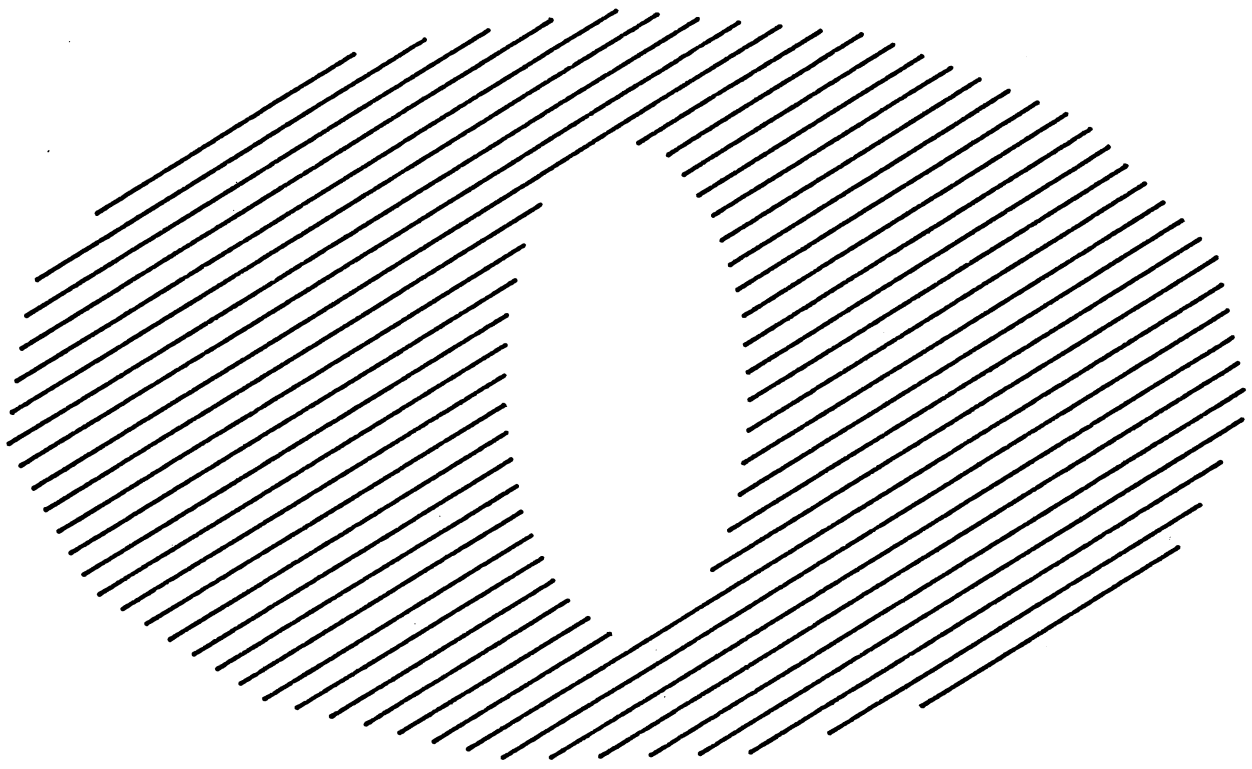
Ellipses are just misshapen circles.

```
DRAW A←(15 POLY 1) IAS 2 2⍴1 2 5 3
```



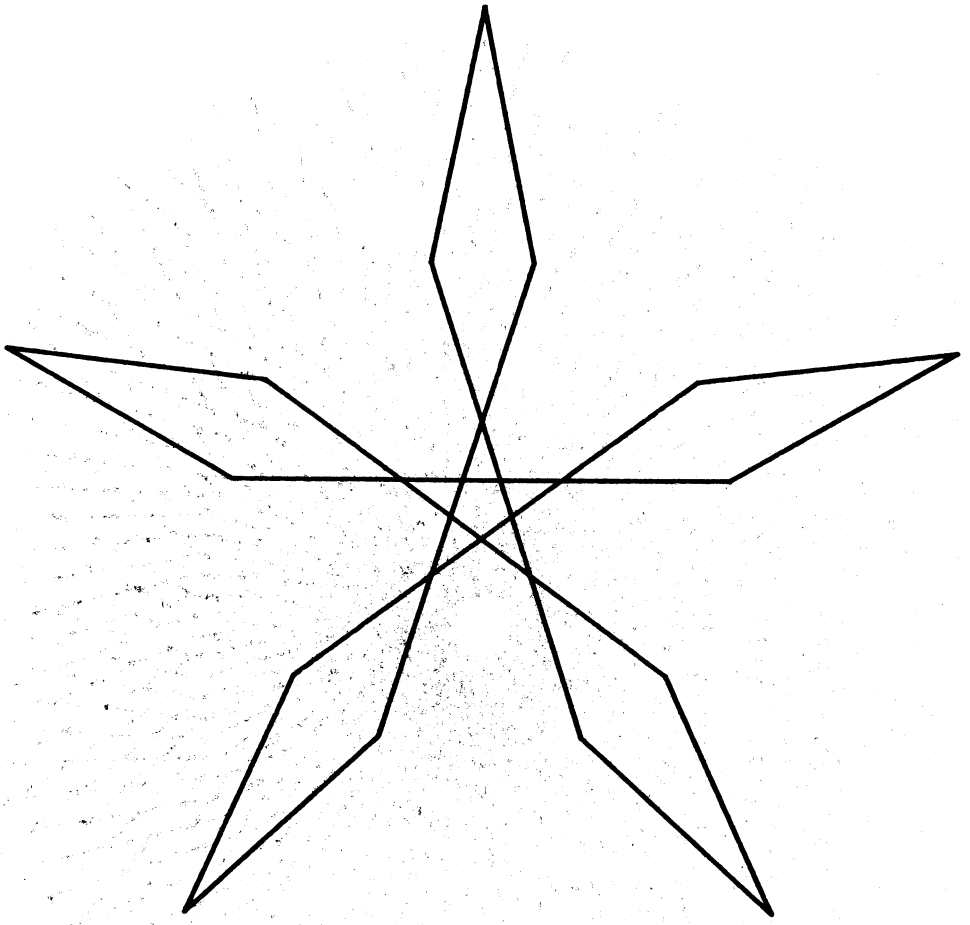
Hatching eggs is trivial.

```
DRAW 30 .2 HATCH A
```



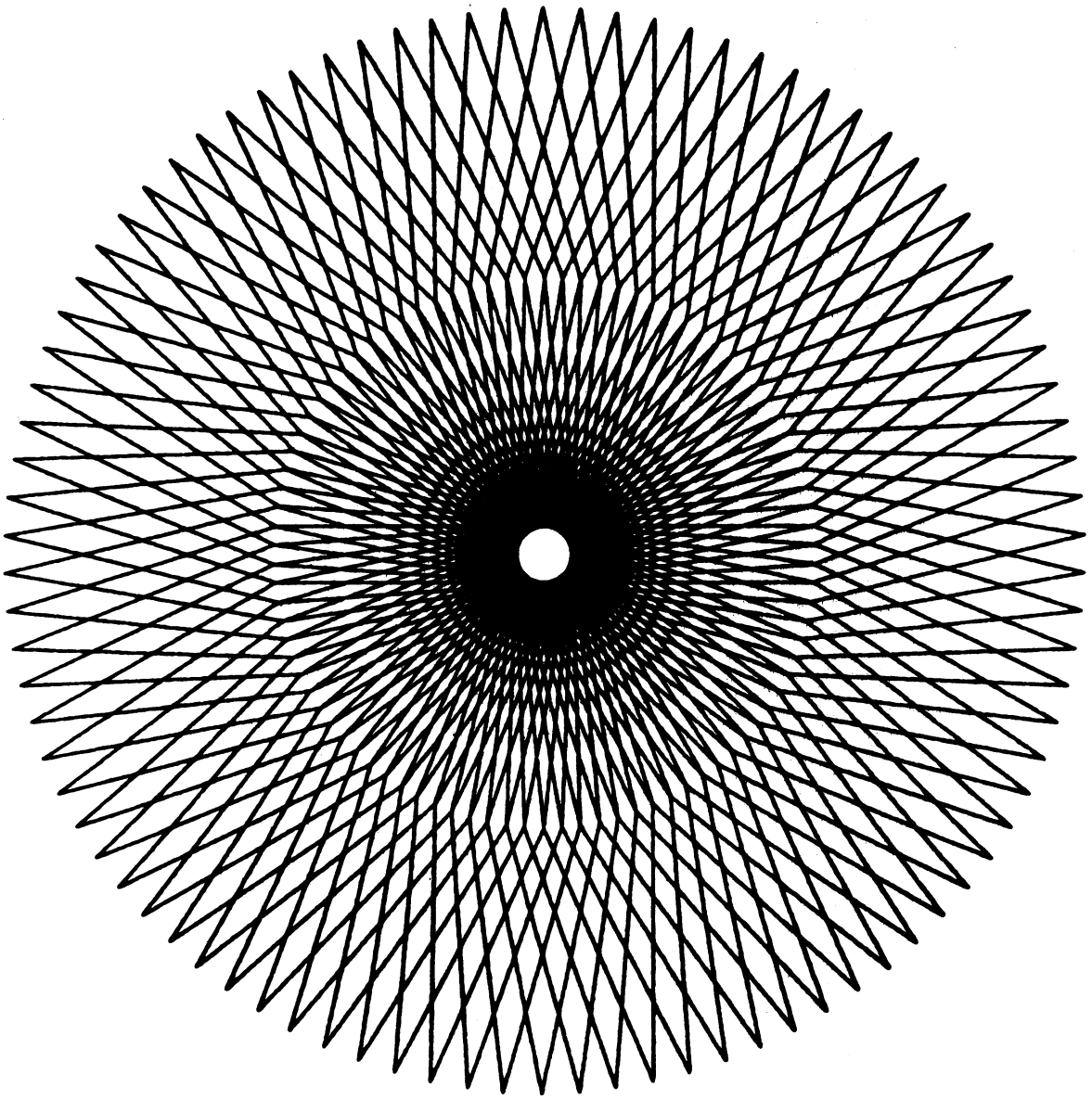
Mandalas.

```
DRAW A←15 5 DAISY 7
```



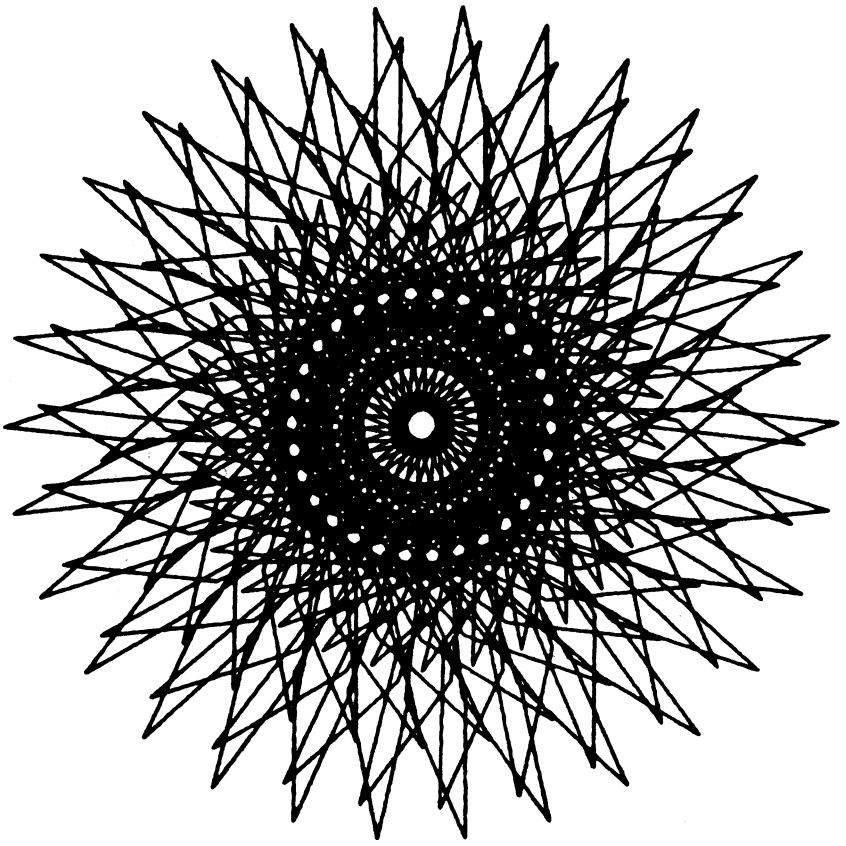
Object A (the daisy above), is rotated.

```
DRAW A IAR 4×ι18
```



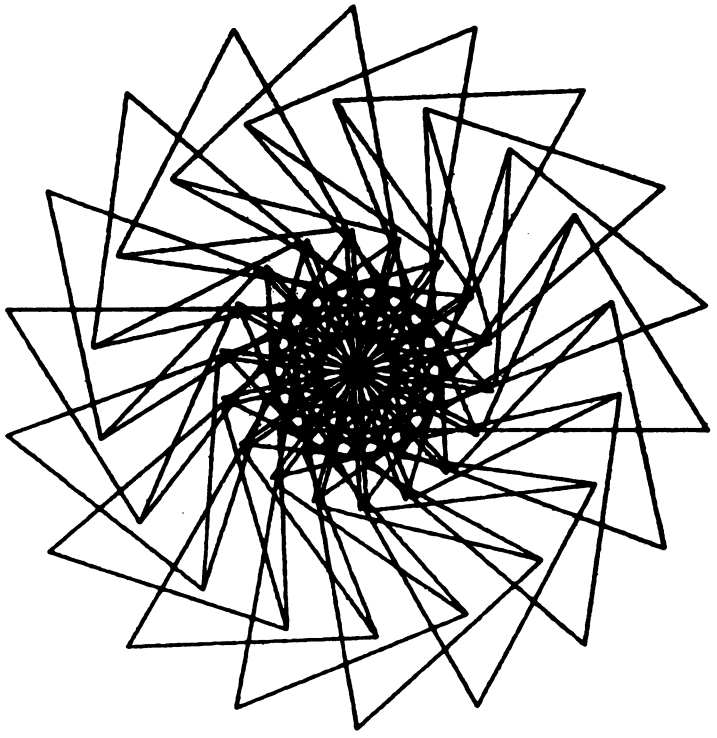
Translation first, then rotation.

```
DRAW (A IAX 3) IAR 12×⍳30
```



Almost any shape rotated about the origin will form a more or less pleasing mandala. Try:

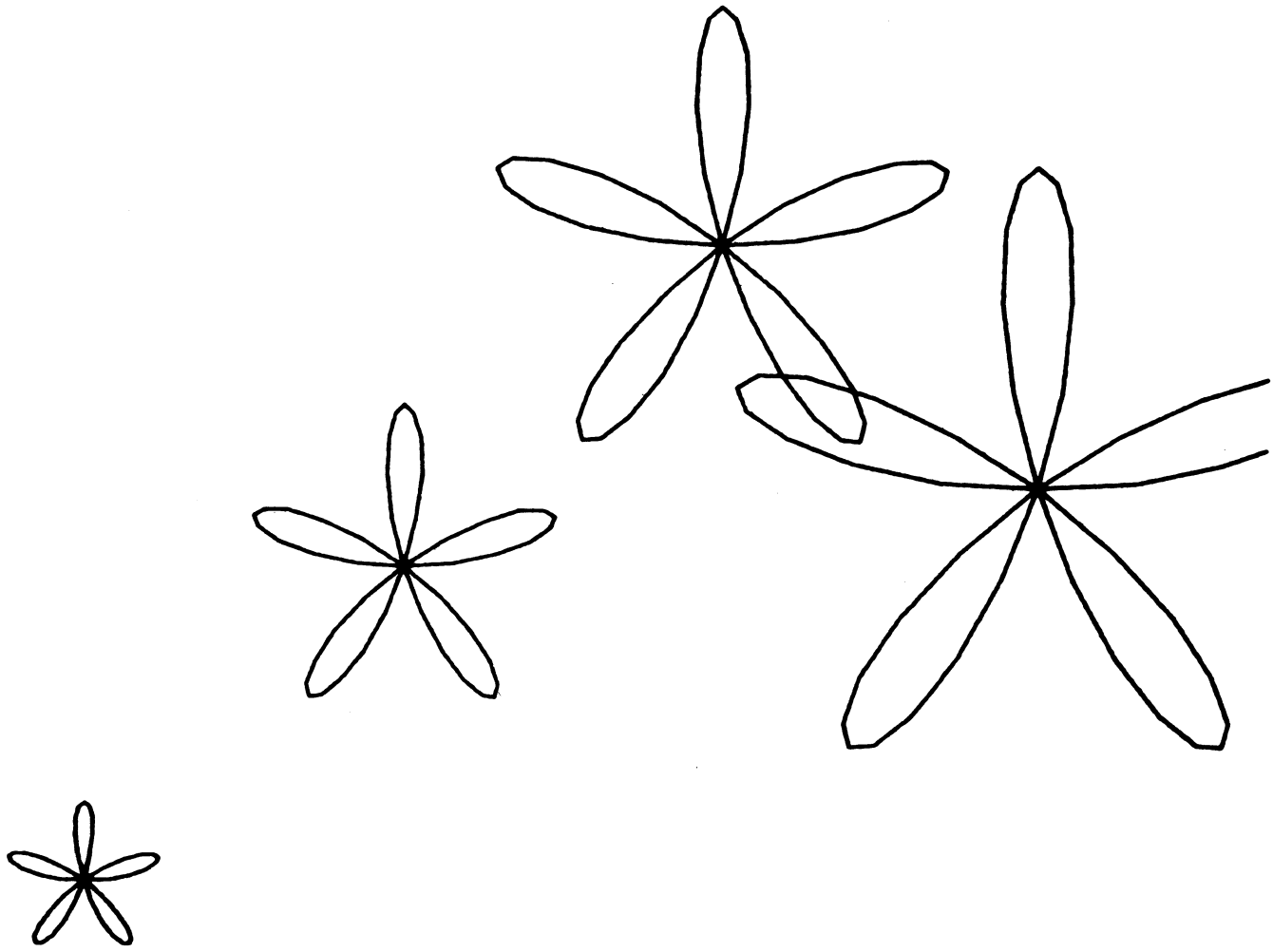
```
DRAW A←FOLLOW
DRAW A IAR 20×⍳18
```



SESSION 2

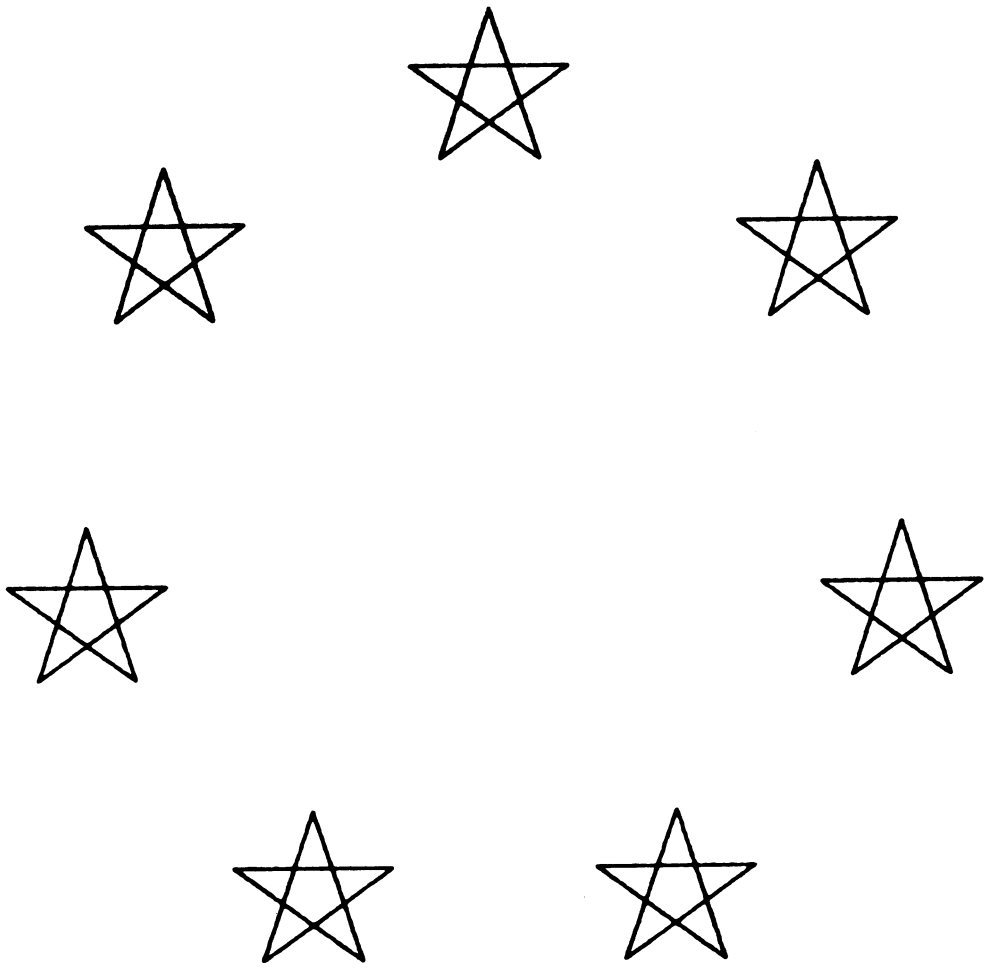
Draw a picture at a series of points *X*.

```
DRAW (50 5 DAISY ι4) IAX 4 2ρ ¯5 ¯5 ¯1 ¯1 3 3 7 0
```



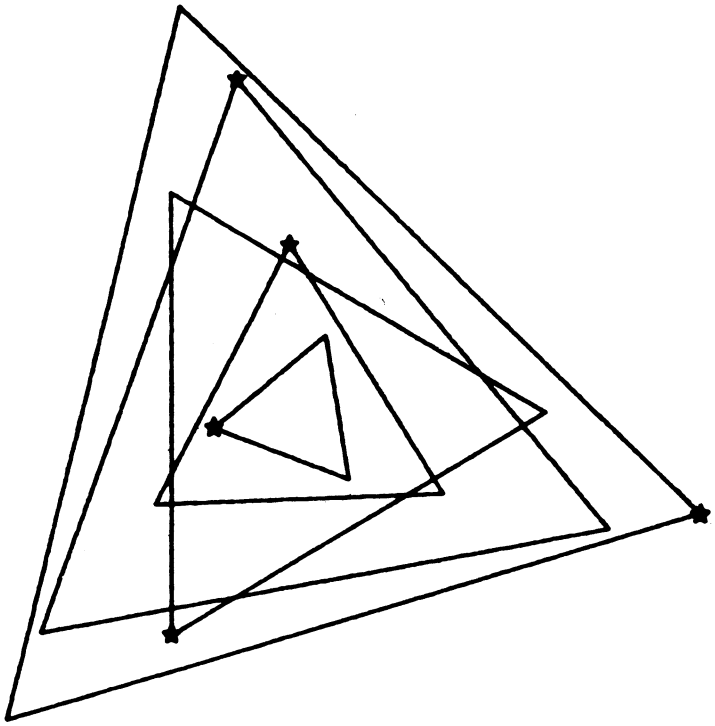
Draw a picture at each point of another picture.

```
DRAW (5 DAISY 1) IAX 0 1↓7 DAISY 5
```



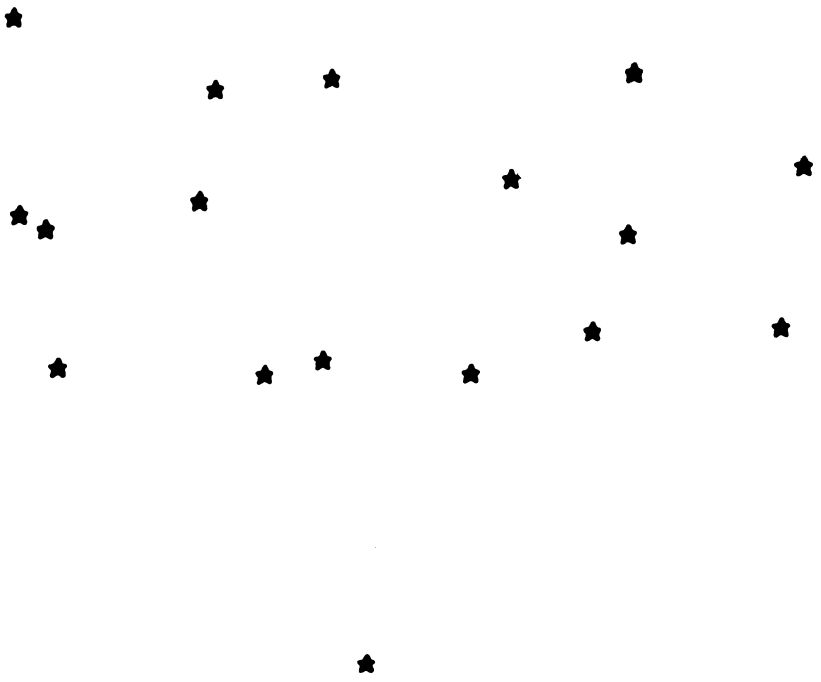
Draw a picture at the first points of another set of pictures:

```
A←(3 POLY ⍳5) IAR 5?360
DRAW (5 6 DAISY .1) IAX A[BEGN A;2 3] ◇ DRAW A
```



Place a star at all points (100 maximum) specified by the crosshairs on the Tektronix terminal using the function *XIN*.

```
DRAW (5 5 DAISY .1) IAX 0 1↓'01' XIN 100
```



SECTION II

TEXTBOOK

Detailed descriptions of all the functions, their syntax, and examples.

KEY TO ABBREVIATIONS

GV	a graphic vector i.e. zero or more objects
TRF	zero or more transforms
XL	translation amounts
SCL	scaling amounts
RU	rotational units
N	numeric data
CH	character data
SV	text string vector (substrings delimited by `1↑`*SV*)
XY	cartesian data
B	Boolean partition vector, (ones start each partition); or Boolean compression/expansion vector
F	file tie number, component-number vector
FNAME	file name
BX	zero or more boxes!s

SECTION II-A

INPUT-OUTPUT FUNCTIONS

GRAPHIC INPUT

(a) Make Object

FUNCTION: **MO**

SYNTAX: *GV←MO XY*

DESCRIPTION: Generates an object from the numeric data *XY*. *MO* is handy for plotting (see page 10).

RIGHT PARAMETER *XY*:

- If *XY* is a scalar, vector, or a 1-column matrix, (⍳⍴,*XY*) will be catenated as *X* coordinates, with *XY* becoming *Y* coordinates. Appropriate controls are added to draw all points.

- If 2≤⍴⍴*XY*, *XY* is restructured to be (2↑(⍴*XY*),1 1)⍴*XY*. Appropriate controls are added to draw all points.

EXAMPLES:

```
      MO 3.1
¯4  1  3.1

      MO .1+⍳3
¯4  1  1.1
 1  2  2.1
 1  3  3.1

      MO 3 2⍴0.1+⍳6
¯4  1.1  2.1
 1  3.1  4.1
 1  5.1  6.1

      MO 2 3⍴.1+⍳6
¯4  1.1  2.1  3.1
 1  4.1  5.1  6.1
```

(b) Polygons

FUNCTION: **POLY**

SYNTAX: *GV←N1 POLY N2*

DESCRIPTION: Generates polygons.

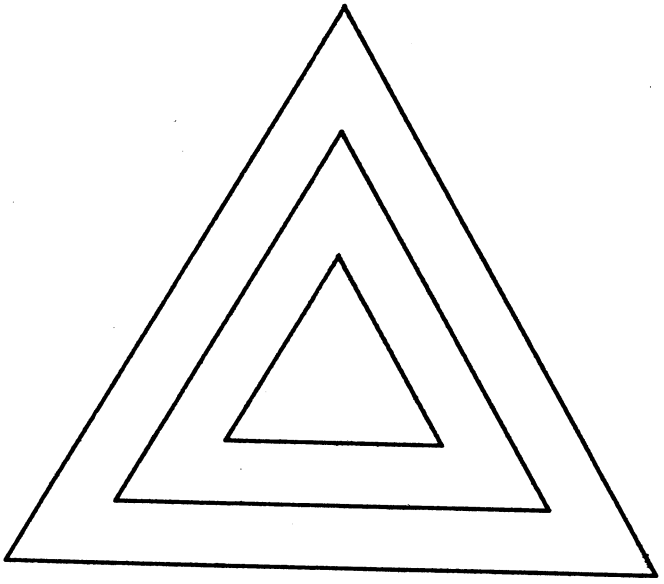
LEFT PARAMETER *N1*: Number of sides to the polygons.

RIGHT PARAMETER *N2*: Radii of the polygons.

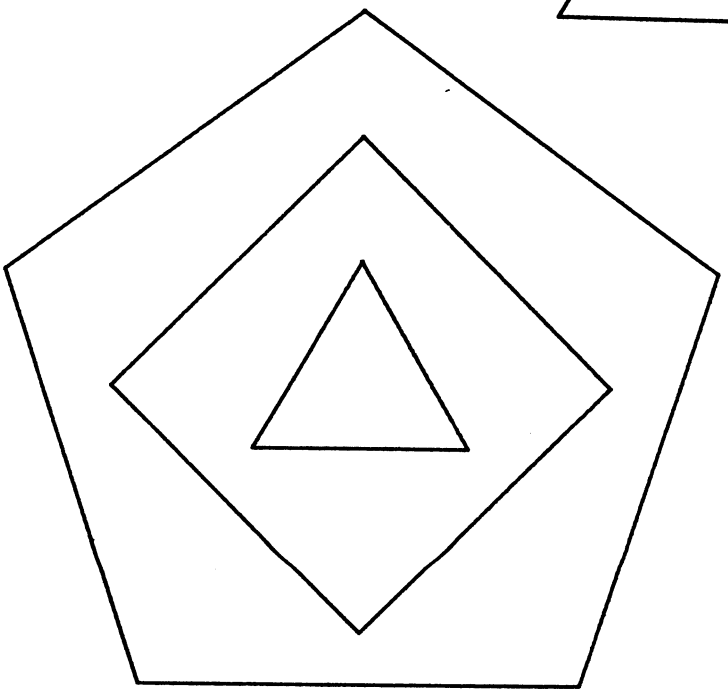
NOTE: *(SHAPE GV)=(ρN1)⌈ρN2*

EXAMPLES:

```
DRAW 3 POLY ⍳3
```



```
DRAW 3 4 5 POLY ⍳3
```



(c) **Daisies and stars**

FUNCTION: **DAISY**

SYNTAX: *GV←N1 DAISY N2*

DESCRIPTION: Generates daisies and stars.

LEFT PARAMETER *N1*: Number of points and number of petals. *N1* may be a scalar, a 2-element vector, a one-column matrix, or a two-column matrix.

- If *N1* is a scalar, *DAISY* generates one star with *N1* points.

- If *N1* is a two-element vector, the first element specifies the number of points that should be used to draw the daisy. The more points, the smoother the curve. The second element defines the number of petals.

- If *N1* is a matrix, multiple stars and daisies are generated:

- One-column matrix: *N[I;]* is the number of points on the I'th star - and must be odd-numbered.

- Two-column matrix: 1↑⍴*N1* is the number of daisies to be built.

N[I;1] is the number of points on the I'th daisy,
N[I;2] is the number of petals on the I'th daisy.

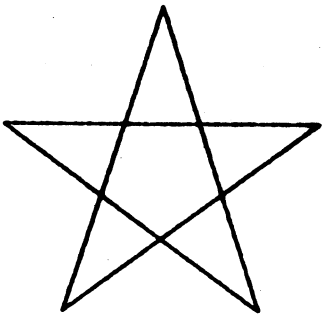
If both values are the same, an *N1[I;2]* pointed star is built.

RIGHT PARAMETER *N2*: Radius of the daisy. *N2* may be a scalar or a vector. When building more than one daisy, *N2[I]* is the radius of the I'th daisy.

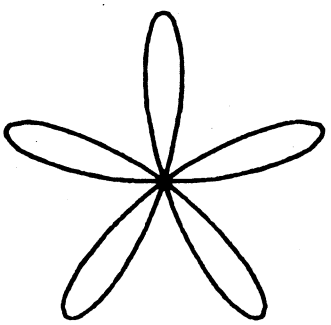
NOTE: *(SHAPE GV)=(⍴N2)⌈1↑⍴N1*

EXAMPLES:

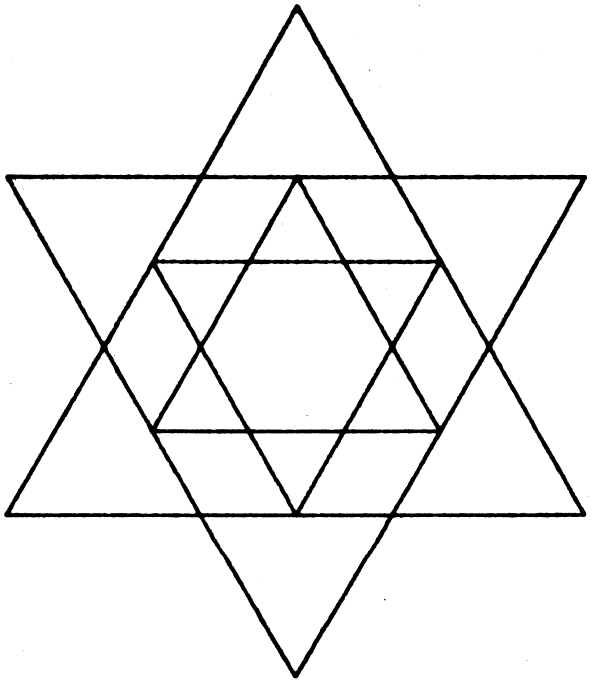
DRAW 5 *DAISY* 2



DRAW 100 5 *DAISY* 2



DRAW 6 *DAISY* 2 4



(d) Interactive input

(d) i Crosshair input is supported only on certain terminals, (the Tektronix 4013/4015 and the Hewlett/Packard 2648A).

FUNCTION: **XIN**

SYNTAX: *GV←CH XIN N*

DESCRIPTION: Crosshair input on the Tektronix 4000 series and the Hewlett/Packard 2648A.

LEFT PARAMETER *CH*: Acceptable input characters.

RIGHT PARAMETER *N*: The maximum number of points allowed to complete the picture (0 is no limit).

NOTES:

1. *XIN* accepts up to *N* points of input from the terminal. The resulting graphic vector is a set of 2-dimensional objects. The value of the controls are determined by the index of the characters (zero origin) in *CH*; resulting controls are:
CONTROLS←¯1+CH⍳INPUT∆CHAR

2. Blank terminates input prematurely.
Carriage return, break, or other ASCII control characters give unpredictable results.

EXAMPLES:

```
'NOLC' XIN 20
```

Creates a graphic vector with a maximum of 20 points. Acceptable input characters are *N*, *O*, *L* and *C*. The value of a control generated by an input character is determined by its origin 0 index in *CH*. Thus, the controls produced by *N*, *P*, *L* and *C* are 0, 1, 2 and 3 respectively.

(d) ii FUNCTION: **FOLLOW**

SYNTAX: *GV←FOLLOW*

DESCRIPTION: "Draw as you go" form of XIN.

NOTES: Input control characters on the Tektronix keyboard are:

Numerics used directly as controls, i.e.:
`0,2,4,6,8` = Move to crosshairs
`1,3,5,7,9` = Draw to crosshairs (from previous point).
N Begin new object (i.e. control is `1⍴OD`)
P Draw to nearest point
L Draw to nearest line
C Circular arc through previous, current and next points.

All other characters are ignored.

EXAMPLES:

A←FOLLOW Crosshairs appear on the screen. Move the crosshairs and key in control characters as necessary to register points.

(d) iii Graphics tablet

FUNCTION: **TABLET**

SYNTAX: *GV←TABLET N*

DESCRIPTION: Pen input on the Tektronix 4953.

RIGHT PARAMETER *N*:

If *N*=0: Point to point input (connect the dots); each push of the pen inputs **one** point. Removing the pen from the tablet terminates input. The result is a single object, containing as many points as there are pen pushes.

If *N*=1: Continuous input (curve or feature tracing). Points are input at several points per second for as long as the pen is pushed. Each pen push produces an object delimiter, allowing several features to be entered at once. The result contains as many objects as there are pen pushes.

OUTPUT FUNCTIONS

(a) Graphic Output

FUNCTION: **DRAW**

SYNTAX: *DRAW GV*

DESCRIPTION: Draws graphic objects.

RIGHT PARAMETER *GV*: If the graphic vector to be drawn is an Nx3 matrix, it is clipped to the screen boundaries and drawn.

Drawing from Files

SYNTAX: *DRAW F*

RIGHT PARAMETER *F*: If *F* is a vector, read and draw components `1↓F` from the file tied to `1↑F`.

If *F* is a scalar or a one-element vector, read and draw all components of the file tied to *F* (character components and packages are ignored).

NOTES: *DRAW* applies the global transform *TRANS*. This maps the user's coordinate space to an origin centered, unit-radius space; clips the picture to the screen boundary as defined by *SMTRANS*; and applies the transform *SMTRANS* to obtain screen coordinates. It then calls the function *OUTTERM* to convert the controls and coordinates into terminal-dependent format, and `⎕`*ARBOUT*s this to the terminal.

(b) **Text Output**. The major output functions for character text output are DRAWSTR and TEXT. Text is defined as a vector of character strings delimited by their first element. The first character after each definition is the type font to use. The shape of the text vector is the number of strings in it. See APPENDIX A (GLOSSARY) - page 120 - for a precise definition of text.

```
     ∇R←SHAPE S
[1]    R←+/STRING∈1↑STRING
     ∇
```

EXAMPLES:

```
      SHAPE '/0FIRST/8SECOND/2THIRD'
3

      SHAPE ' THIS HAS FIVE INVALID STRINGS'
5

      SHAPE ''
0
```

The set of APL-like functions for graphic vectors also manipulate character-string vectors.

FUNCTION: **DRAWSTR**

SYNTAX: *BX DRAWSTR SV*

LEFT PARAMETER *BX*: Zero or more boxes to set the boundaries for the text to be drawn. For example, *BX* may be a rectangle with the bottom left-hand corner at screen coordinates (0,0) and the top right-hand corner at the point (1,5). *BX* would be defined as follows:
BX←0 0 1 5

RIGHT PARAMETER *SV*: Text string vector. Strings are delimited by the first character in the first string. The first character after each string delimiter in *SV* is the type font.

On all terminals, font 0 is APL hardware characters and fonts 8 and 9 are stroke characters (drawn representations of the standard character set).
On the *TEK*4015 type fonts 0, 1, 2 and 3 are APL; 4, 5, 6 and 7 are ASCII;
(4|type) controls character size.

Each string in *SV* is drawn such that the first character in the string is located at the upper left-hand corner of the box. Stroke characters are scaled to fit the box.

NOTE: The following condition must hold:
(1↑⍴*BX*)=*SHAPE SV*

EXAMPLES:

```
(3 4⍴0 0 5 1) DRAWSTR '/0ONE/8TWO/2THREE'
```

FUNCTION: **TEXT**

SYNTAX: *BX TEXT SV*

DESCRIPTION: Similar to DRAWSTR but does not support stroke characters.

EXAMPLES:

```
(3 4ρ0 0 5 1) TEXT '/0ONE/8TWO/2THREE'
```

FUNCTION: **VTEXT**

SYNTAX: *GV←VTEXT SV*

DESCRIPTION: Inserts linefeeds and backspaces for vertical text.

EXAMPLES:

```
(1 4ρ0 0 1 5) DRAWSTR VTEXT '/8VERTICAL TEXT'
```

FUNCTION: **STR**

SYNTAX: *GV←STR SV*

DESCRIPTION: Produces stroke (drawn) characters.

RIGHT PARAMETER *SV*: Text string vector. One object is created for each character in *SV* (excluding delimiters and type fonts).

EXAMPLES:

```
      A←'/8EENY MEANY/8MINEY MOE'
      SHAPE A
2
      A←STR A
      SHAPE A
19
      DRAW A INBOX 0 0 5 1
```

FUNCTION: **STR5**

SYNTAX: *GV←STR5 SV*

DESCRIPTION: Produces stroke (drawn) characters.

RIGHT PARAMETER *SV*: Text string vector. One object is created for each substring in *SV*.

EXAMPLES:

```
      A←'/8EENY MEANY /8MINEY MOE'
      SHAPE A
2
      A←STR5 A
      SHAPE A
2
      (2 4ρ0 0  5 1    5 1  8 2) DRAWSTR A
```

MINEY MOE

EENY MEANY

(c) **Boxes and Clipping**. Boxes may be used for placement of text or for moving pictures about in a manner akin to "viewport/window" techniques.

A box is an Nx4 matrix representing a collection of N rectangles. In standard format, each row of the matrix represents the lower left cartesian (X,Y) coordinate and the upper right cartesian (X,Y) coordinate of the rectangle. Interesting effects can be produced by interchanging box coordinates.

FUNCTION: **BOX**

SYNTAX: *GV←BOX BX*

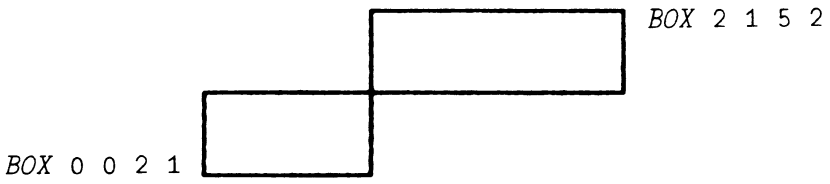
DESCRIPTION: Produces a drawable graphic vector of the boxes specified by *BX*.

RIGHT PARAMETER *BX*: *BX* defines the lower left and upper right corners of the boxes.

EXAMPLES:

```
DRAW  BOX 0 0 2 1
```

```
DRAW BOX 2 1 5 2
```



FUNCTION: **INBOX**

SYNTAX: *GVR←GV INBOX BX*

DESCRIPTION: Places *GV* inside the boxes *BX*.

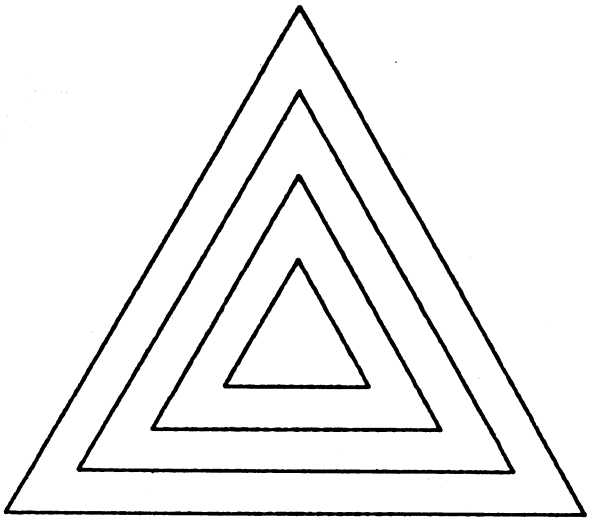
LEFT PARAMETER *GV*: A graphic vector.

RIGHT PARAMETER *BX*: An Nx4 matrix of boxes. Each row is a box.

NOTE: *(SHAPE GVR)=(SHAPE GV)⌈1↑⍴BX*
The shorter argument is extended using *RESHAPE*. The *I*'th object in *GV* is placed in the *I*'th box.

EXAMPLES:

```
DRAW A←3 POLY ⍳4
```



```
DRAW BOX 0 0 2 1
```

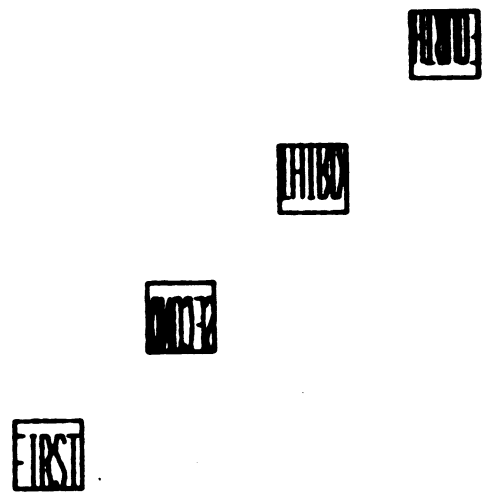
```
DRAW A INBOX 0 0 2 1
```

Swapping box coordinates allows mirror images and reflections:

```
A←.3×4 4⍴1 2 3 4   7 6 5 8   9 12 11 10   15 16 13 14
```

Above boxes are: normal, swap X, swap Y, swap X and Y

```
TXT←'⍬FIRST⍬SECOND⍬THIRD⍬FOURTH'
DRAW (STR 5 TXT) INBOX A
DRAW BOX A
```



FUNCTION: **CLIP**

SYNTAX: *GVR←BX CLIP GV*

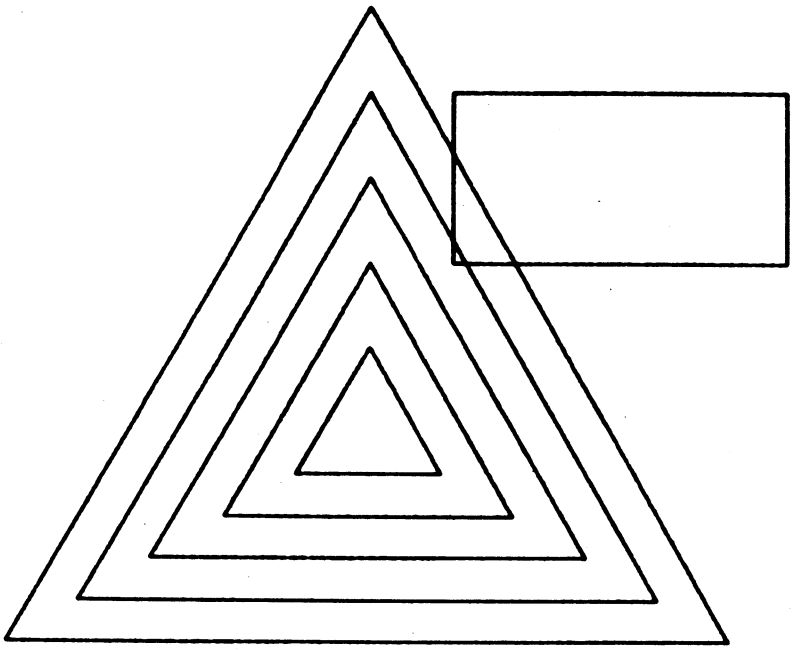
DESCRIPTION: Rectangular clipping - *GV* is clipped to the box *BX*.

LEFT PARAMETER *BX*: A box that defines the clipping boundary.

RIGHT PARAMETER *GV*: Vector of objects to be clipped by *BX*.

EXAMPLES:

```
DRAW 3 POLY ⍳5
```



```
DRAW BOX 1 2 5 4
```

```
DRAW 1 2 5 4 CLIP 3 POLY ⍳5
```

FUNCTION: **IMM**

SYNTAX: *BX←IMM GV*

DESCRIPTION: Itemwise minimum/maximum of the graphic objects in *GV*.

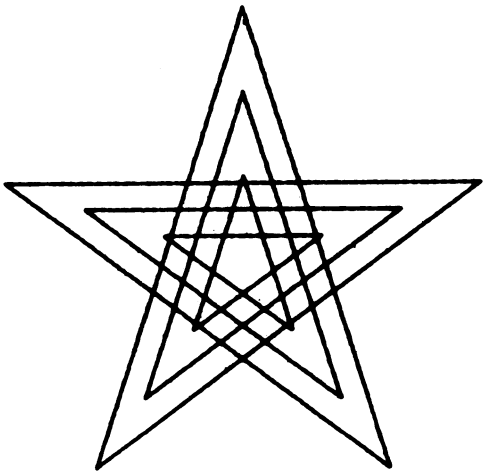
RIGHT PARAMETER *GV*: Graphic vector.

RESULT *BX*: The result is boxes that fit exactly around the objects *GV* - i.e. the bottom left corner coordinates are calculated for the minimum X,Y, the top right-hand corner for the maximum X,Y of the objects in *GV*.

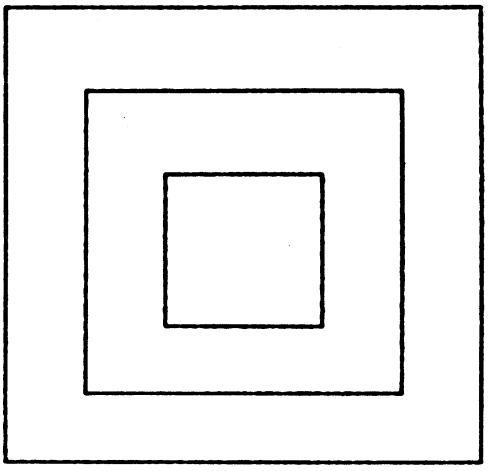
NOTE: `(1↑⍴BX)=SHAPE GV`

EXAMPLES:

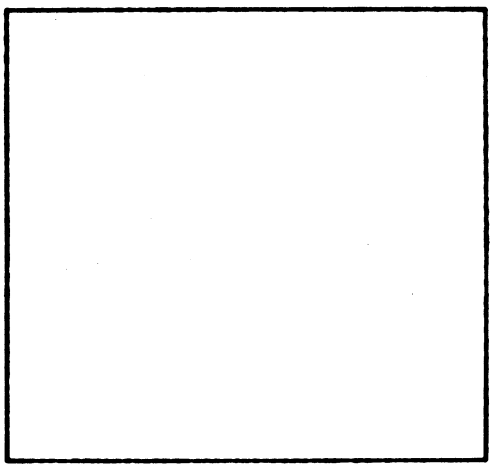
```
DRAW A←5 DAISY ⍳3
```



DRAW BOX IMM A



DRAW BOX IMM MOS A



(d) **Grid functions** - GRID and MENU.

FUNCTION: **GRID**

SYNTAX: *GV←GRID N*

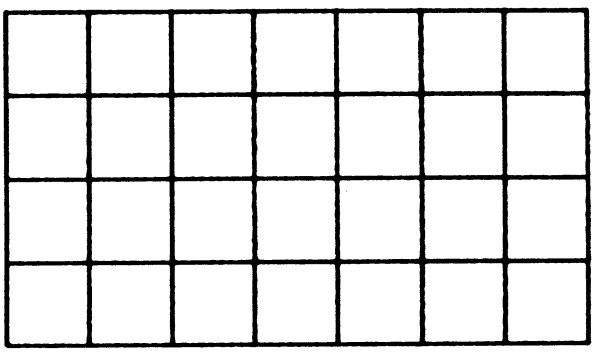
DESCRIPTION: Generates a line grid.

RIGHT PARAMETER *N*:

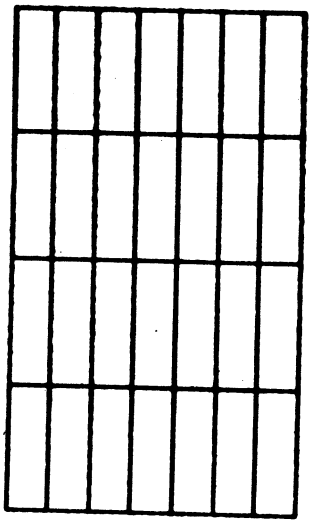
N[1 2] = grid size in elements, X by Y
N[3 4] = interelement spacing (X,Y)
(default 1,1)
N[5 6] = lower left corner of the grid,
(default 0 0).

EXAMPLES:

```
DRAW GRID 7 4
```



DRAW GRID 7 9 .5 1.5



FUNCTION: **MENU**

SYNTAX: *R←N MENU CH*

DESCRIPTION: Select from a menu with crosshairs. This function is supported on the Tektronix 4013/4015 and the HP 2648A.

LEFT PARAMETER: *N* is the maximum number of selections allowed. Zero implies no limit.

RIGHT PARAMETER: *CH* is a character matrix of menu items.

RESULT *R*: Is a vector of origin 1 row indices into the character matrix of the items selected. Each index is produced by user-selection from the menu.

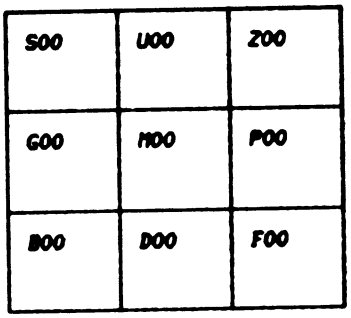
NOTES: The menu is drawn and the crosshairs appear. The user selects from the menu by placing the crosshairs inside the rectangle containing the name of the desired item, and striking any graphic key. A star will be drawn in the box indicating that a selection was made. A space will terminate the menu prematurely.

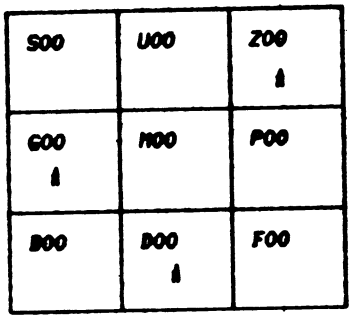
EXAMPLES:

```
      DIR←9 3⍴'BOODOOFOOGOOMOOPOOSOOUOOZOO'
      DIR
BOO
DOO
FOO
GOO
MOO
POO
SOO
UOO
ZOO
```

```
      CHOICES←0 MENU DIR
```

Here *ZOO*, *GOO*, *DOO* were selected (in that order).






```
      CHOICES
9  2  4
```

(e) Screen Control

FUNCTION: **COMIX**

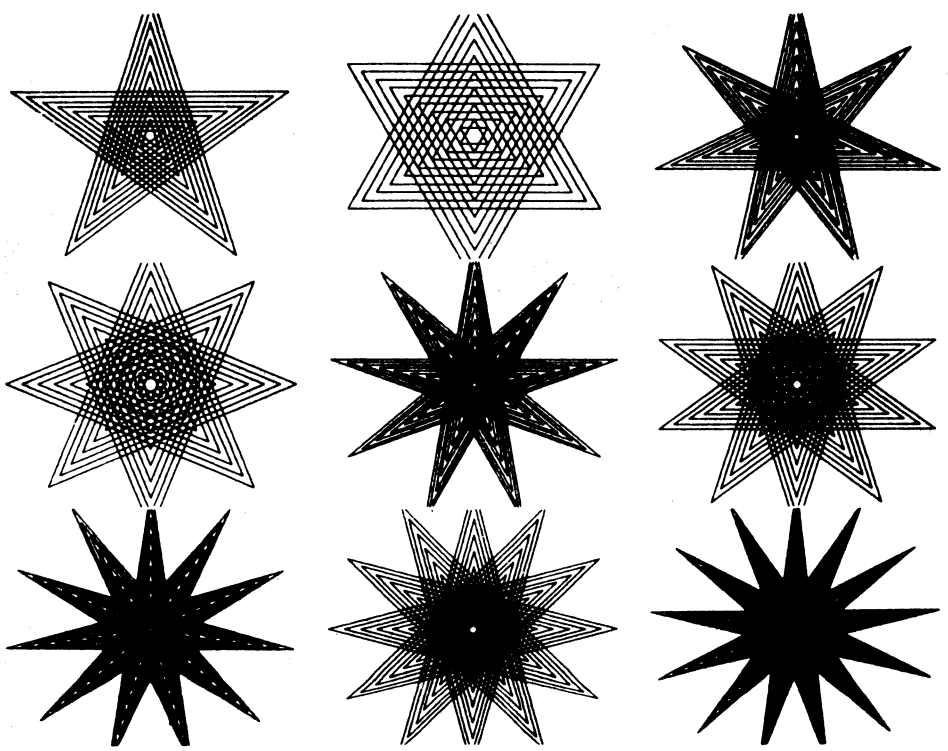
SYNTAX: *TRF←COMIX N*

DESCRIPTION: Produces comic strip transforms. The result is normally assigned to the global variable *CTRANS* which is used to divide the terminal screen into several subscreens.

RIGHT PARAMETER *N*: *COMIX* will produce *N*×2 transforms.

NOTE: *COMIX* is a transform function, and is listed as such in APPENDIX B.

```
    ∇FOO N;I;CTRANS
[1]   CTRANS←COMIX N
[2]   N←N×N ◇ I←1 ◇ ERASE
[3]  L0: DRAW (4+I) DAISY ⍳9  ◇  NEXTFR
[4]   →(N≥I←I+1)⍴L0
    ∇

      FOO 3
```



FUNCTION: **FRAME**

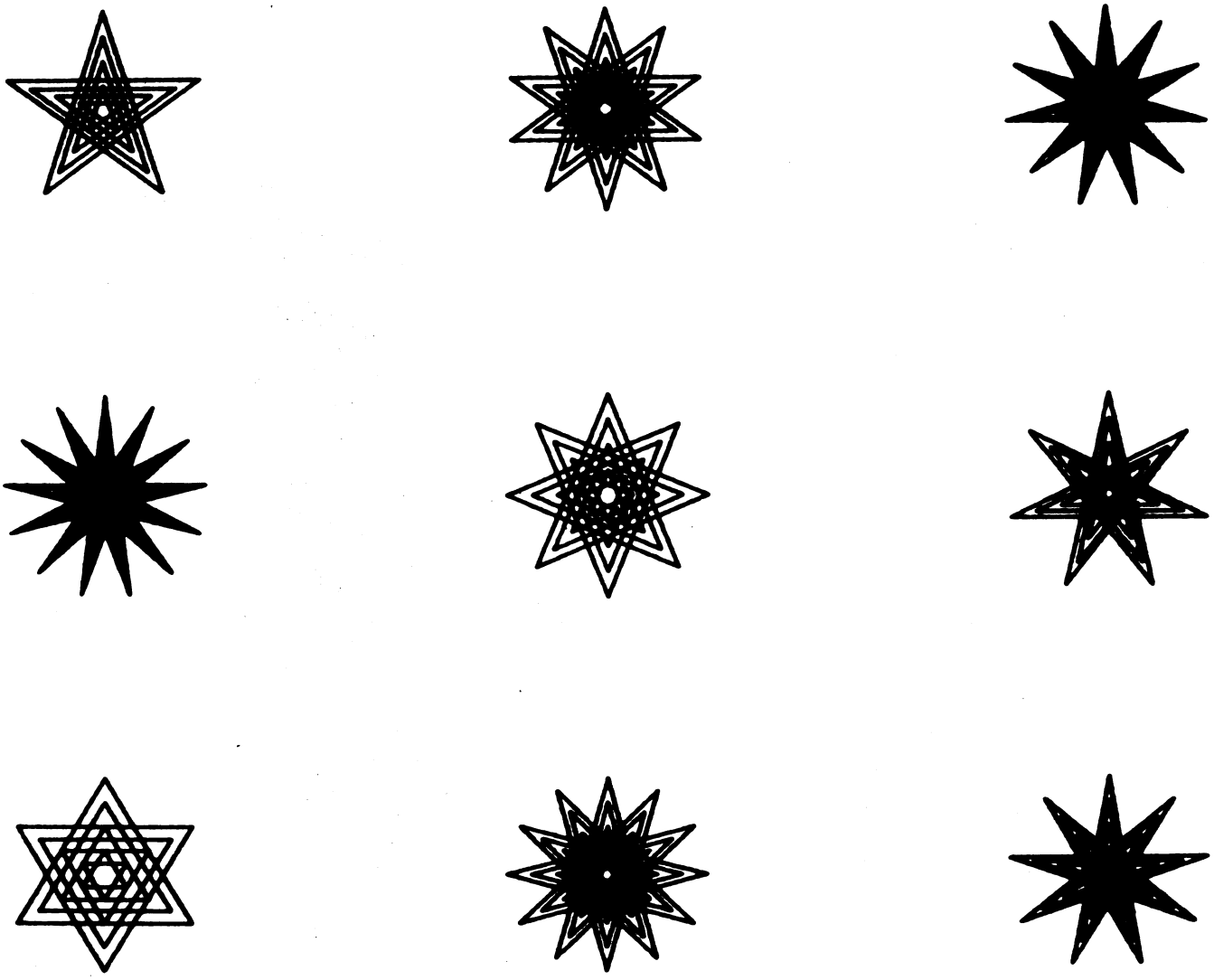
SYNTAX: *FRAME N*

DESCRIPTION: Causes frame (1↑ρ*CTRANS*)|*N* to be used for the next draw.

RIGHT PARAMETER *N*: The frame number to use for the next draw.

EXAMPLES:

```
    ∇FOO N;I;J
[1]   CTRANS←COMIX N
[2]   N←N×N ◇ J←N?N ◇ I←1
[3]  L0: FRAME J[I]
[4]   DRAW (4+I) DAISY ⍳4
[5]   →(N≥I←I+1)ρL0
    ∇

      FOO 3
```



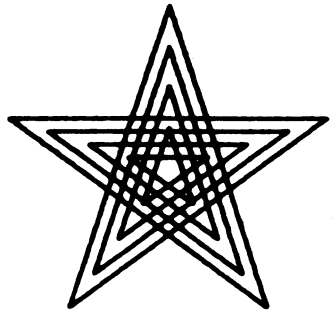
FUNCTION: **NEXTFR**

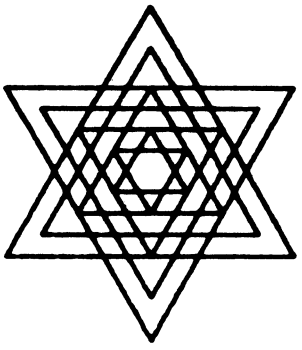
SYNTAX: *NEXTFR*

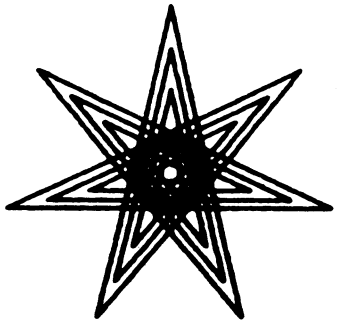
DESCRIPTION: Causes the next subscreen to be used for subsequent *DRAW*s. When all subscreens have been used, a *WAITCR* ◇ *ERASE* is executed.

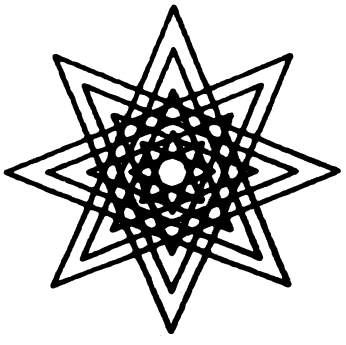
EXAMPLES:

```
   ∇FOO N;I
[1]    CTRANS←COMIX N
[2]    N←N×N ◇ I←1 ◇ ERASE
[3]  L0: DRAW (4+I) DAISY ⍳4
[4]    NEXTFR ◇ →(N≥I←I+1)⍴L0
     ∇

       FOO 2
```









FUNCTION: **ERASE**

SYNTAX: *ERASE*

DESCRIPTION: Clears the screen and homes the cursor on CRT terminals; on hard copy terminals a form feed is executed.

NOTES: ERASE has no effect on **Multiwriter II** terminals, as these ignore form feed.

FUNCTION: **WAITCR**

SYNTAX: *WAITCR*

DESCRIPTION: Waits for a carriage return.

FUNCTION: **WHATFR**

SYNTAX: *N←WHATFR*

DESCRIPTION: Returns the current frame number.

(f) **Square Plots.**

FUNCTION: **SQPLOT**

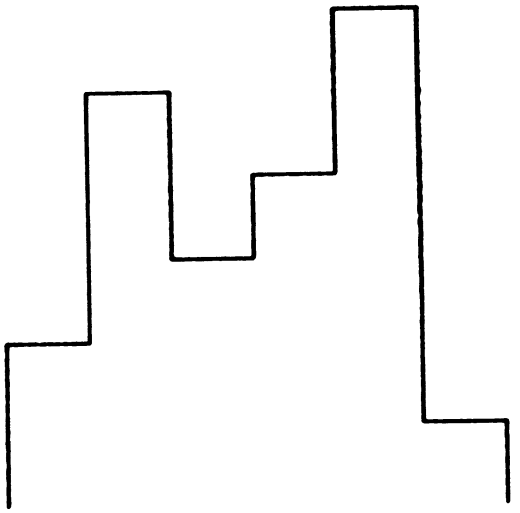
SYNTAX: *GV←SQPLOT N*

DESCRIPTION: Creates a graphic vector to draw a square plot.

RIGHT PARAMETER *N*: *N* is a numeric vector of data to be plotted.

EXAMPLES:

```
DRAW SQPLOT 6?6
```



(g) **The "In between" functions** INBT and DRAWINBT.

FUNCTION: **INBT**

SYNTAX: *GV3←N INBT GV1,[1]GV2*

DESCRIPTION: Generate a set of *N* objects between two graphic vectors *GV1* and *GV2* by linear interpolation.

LEFT PARAMETER *N*: *N* is the number of objects to be generated.

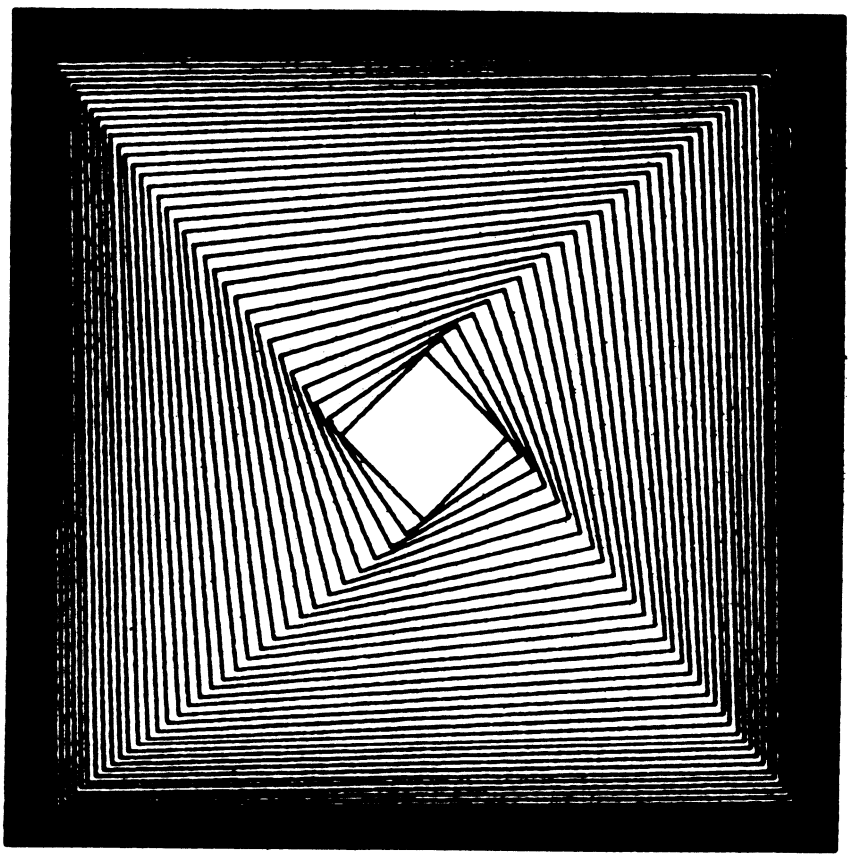
FUNCTION: **DRAWINBT**

SYNTAX: *N DRAWINBT GV*

DESCRIPTION: The same as *DRAW N INBT GV* but avoids *WS FULL* by drawing as it goes.

EXAMPLES:

```
DRAW A←(4 POLY 1),[1]BOX ¯5 ¯5 5 5
DRAW 10 INBT A
```

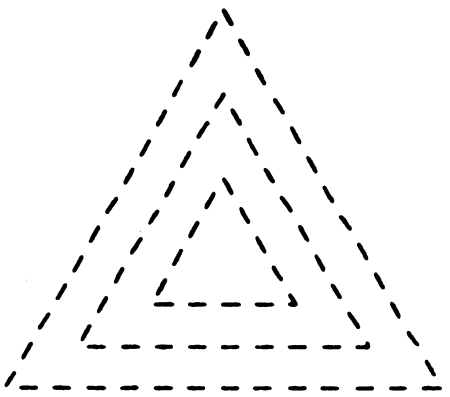


(h) **Decorating** with the HATCH group of functions.

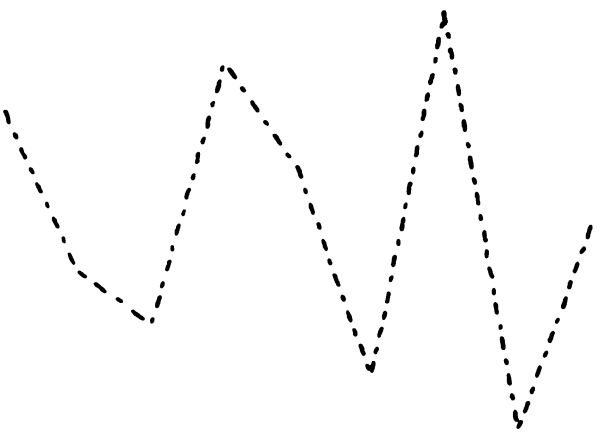
FUNCTION:	**DASH**
SYNTAX:	*GVR←N DASH GV*
DESCRIPTION:	Restructure *GV* to draw with dashed lines.
LEFT PARAMETER *N*:	A matrix of dash lengths and interdash spaces.
RIGHT PARAMETER *GV*:	The graphic vector of objects to be dashed.

EXAMPLES:

```
DRAW  .2  .2  DASH  3 POLY ⍳3
```



```
DRAW  .2  .2  .1  .2  DASH MO -4+9?9
```



FUNCTION: **HATCH**

SYNTAX: *GV←N HATCH GV*

DESCRIPTION: Cross-hatch *GV*.

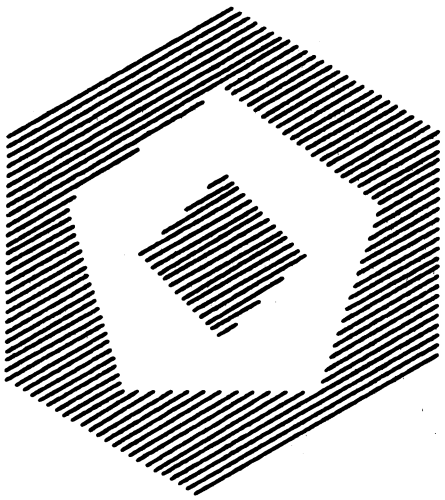
LEFT PARAMETER *N*: *N*[1]=angle of hatch lines
N[2]=spacing between hatch lines.

RIGHT PARAMETER *GV*: Graphic vector of the object to be hatched.

NOTE: *HATCH* will produce unpredictable results if *GV* is not a closed object.

EXAMPLES:

```
DRAW 30 .1 HATCH 4 5 6 POLY ⍳3
```



SECTION II-B

TRANSFORMS

Transforms (*TRF*) are rank-3 arrays that describe the movement and position of objects in space. Each plane represents a single linear transform that may be applied to a single object or to all objects in a graphic vector by use of the dyadic functions *IAT* and *AT* respectively.

For pictures in two dimensions, a single transform is a 1x3x3 array representing values of rotation, translation and scaling, while for three-dimensional objects it is a 1x4x4 array. Transforms may generally be combined by catenation along the first coordinate. Most graphic functions treat a transform as a one-element vector or scalar.

BASIC TRANSFORMS

1) Scaling

FUNCTIONS: SCALE and SAP

DESCRIPTION: The process of shrinking or magnifying an object. Scaling may be considered analogous to "zoom" and may be performed about the coordinate origin using SCALE, or a specified point using SAP.

2) Rotation

FUNCTIONS: ROT and RAP

DESCRIPTION: The process of rotating an object about a point, either about the coordinate origin using ROT, or about a specified point using RAP.

3) Translation

FUNCTION: XLATE

DESCRIPTION: The process of moving objects. The function *XLATE* applied to an object will move it in a manner analogous to "reposition', "tilt" and "pan".

NOTE: The function RSAP may be used to simultaneously rotate and scale about a point.

TRANSFORM GENERATORS

The transform generating functions SCALE, ROT, XLATE and others - see APPENDIX B (page 131) for a complete list, generate arrays that will scale, rotate or translate objects when "applied" to them.

For example, *XLATE X* where *X* is a scalar, will produce a transform that will move an object to the left or right when applied to the object using one of the functions AT or IAT. Similarly, *SCALE S* will produce a transform that will enlarge or shrink an object when applied. More details follow:

FUNCTION: **SCALE**

SYNTAX: *TRF←SCALE S*

DESCRIPTION: SCALE produces transforms to scale about the origin.

RIGHT PARAMETER *S*: The shape of *S* controls scaling as follows:

(a) `1=⍴⍴S`
Scaling in all coordinates by *S* when *S* is a scalar or a one-element vector (no distortion).

(b) *S*=(*R*,1) matrix
Scaling in all coordinates by *S* when *S* is a one-column matrix.

For example:
3
2
1
is treated as (*S*,*S*),
3 3
2 2
1 1
as a convenience.

(c) *S*=(*R*,2) matrix
Scaling in 2-space (two dimensions) along both the X and Y axes when *S* is a two-column matrix. Each row of *S* is a scaling; column 1 is a scaling along the X-axis and column 2 along the Y-axis.

(d) *S*=(*R*,*N*) matrix
Scaling in N-space (3 or more dimensions), by *S*, when *S* is an N-column matrix. Each row of *S* is a scaling.

N must equal the number of dimensions in which you are working.

FUNCTION: **ROT**

SYNTAX: *TRF*←*ROT RU*

DESCRIPTION: ROT produces transforms which rotate objects about the origin.

RIGHT PARAMETER *RU*: *RU* specifies the rotation to be applied to the object. Both the unit and the direction of rotation are dependent on the variable *DEGR* (see page 118). If the default value of *DEGR* is used, rotation is measured in degrees. A positive rotation (*RU*>0) is counter-clockwise.

The shape of *RU* controls rotation as follows:

(a) 1=×/⍴*RU*
One transform is created when *RU* is a scalar or a one-element vector. The transform specifies a rotation of *RU* units.

(b) *RU*=(*N*,1) matrix
When *RU* is an (*N*,1) matrix, *N* transforms are created. That is, *N*=1↑⍴*TRF*. The *I*'th transform *TRF*[*I*;;] specifies a rotation of *R*[*I*;] units.

(c) *RU*=(*N*,2) matrix
A 3-D transform is created for each row of *RU*. The first column specifies the coordinate the rotation is to be applied to; the second column is the rotation to be applied. For example, if *RU*[*I*;]←2 30, the rotation applied will be 30 units about the second coordinate (the Y-axis). Note that *RU*[*I*;1] **must** be either 1, 2, or 3.

(d) *RU*=(*N*,3) matrix
A 3-D transform is created for each row of *RU*. The columns specify the rotation to be applied to the first, second and third coordinates respectively.

FUNCTION: **XLATE**

SYNTAX: *TRF←XLATE X*

DESCRIPTION: XLATE generates translation transforms.

RIGHT PARAMETER *X*: The shape of *X* controls the translation as follows:

(a) `1=×/ρX`

Translation along the first coordinate (X-axis), by *X* units, when *X* is a scalar or a one-element vector.

(b) *X*=(*R*,1) matrix

Translation along the first coordinate by *X* units, when *X* is a one-column matrix which is treated as (*X*,0) as a convenience.
For example:

```
3
2
1
```

is treated as

```
3 0
2 0
1 0
```

(c) *X*=(*R*,2) matrix

Translation in 2-space (two dimensions) along both the X and Y-axes. Each row of *X* is a translation; column 1 is translation along the X-axis, column 2 along Y.

(d) *X*=(*R*,*N*) matrix

Translation in N-space along all coordinates when *X* is an N-column matrix.

N **must** equal the dimensions you are working in:
2-dimensions --> *N*=2
3-dimensions --> *N*=3
etc.

TRANSFORM APPLICATORS

Once the transforms have been generated, we need functions to **apply** them to graphic vectors. These functions are *AT* and *IAT*.

FUNCTION: **AT**

SYNTAX: *GVR←GV AT TRF*

DESCRIPTION: Outerproductly apply transform. Every transform in *TRF* is applied to every object in *GV*.

LEFT PARAMETER *GV*: The graphic vector to be transformed.

RIGHT PARAMETER *TRF*: The transforms that will be applied to *GV* to produce the transformed graphic vector *GVR*.

NOTE: *(SHAPE GVR) = (SHAPE GV)×1↑⍴TRF.*

FUNCTION: **IAT**

SYNTAX: *GVR←GV IAT TRF*

DESCRIPTION: Itemwise apply transform. Each transform in *TRF* is applied to the corresponding object in *GV*. The shorter argument is reused as in *RESHAPE*.

PARAMETERS: The parameters for *IAT* are the same as those for *AT*.

NOTE: *(SHAPE GVR) = (SHAPE GV)⌈1↑⍴TRF.*

COMBINING TRANSFORMS

Transforms may be combined when two or more have to be applied to the same graphic vector. The first transform may be transformed by the second giving a combined transform to be applied to the graphic vector. This helps to prevent *WS FULL* and tends to be more CPU-efficient than applying each transform separately.

For example:

```
(SCALE .2) TT XLATE 5 1
```

will produce a transform that will both scale and translate an object.
The dyadic functions *TT* and *ITT* combine two transforms. The result is a new transform.

FUNCTION: **TT**

SYNTAX: *TRFR←TRFA TT TRFB*

DESCRIPTION: Outerproductly combine transforms. Every transform on the right is applied to every transform on the left, that is:
`(1↑⍴TRFR) = (1↑⍴TRFA)×1↑⍴TRFB`

FUNCTION: **ITT**

SYNTAX: *TRFR←TRFA ITT TRFB*

DESCRIPTION: Itemwise combine transforms. Each transform on the right is applied to the corresponding transform on the left. The shorter argument is reused, as in *RESHAPE*.
`(1↑⍴TRFR) = (1↑⍴TRFA)⌈1↑⍴TRFB`

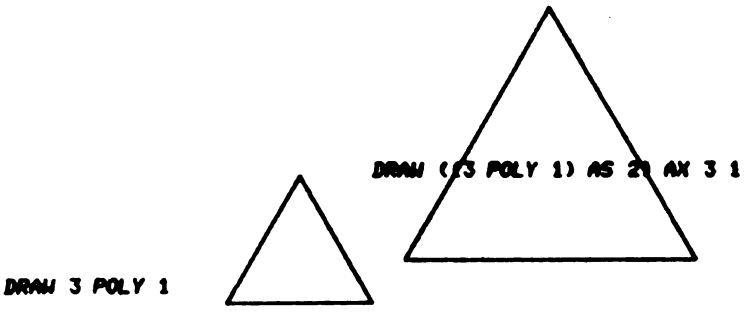
One rather important point which, once understood, avoids utter dark mystery:

When transforms are combined, they act as if they had been applied separately in the order from left to right. Since transform combining is basically a +.× process, it is **not** commutative.

The order in which transforms **must** be combined, is therefore SCALE->ROTATE->TRANSLATE.

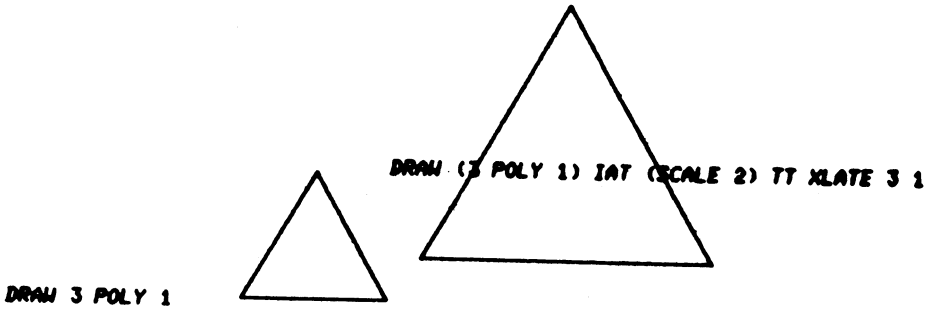
To see why the order in which transforms are combined is important, try these examples:

```
DRAW ((3 POLY 1) AS 2) AX 3 1 ⍝ RIGHT: SCALE, THEN TRANSLATE.
```



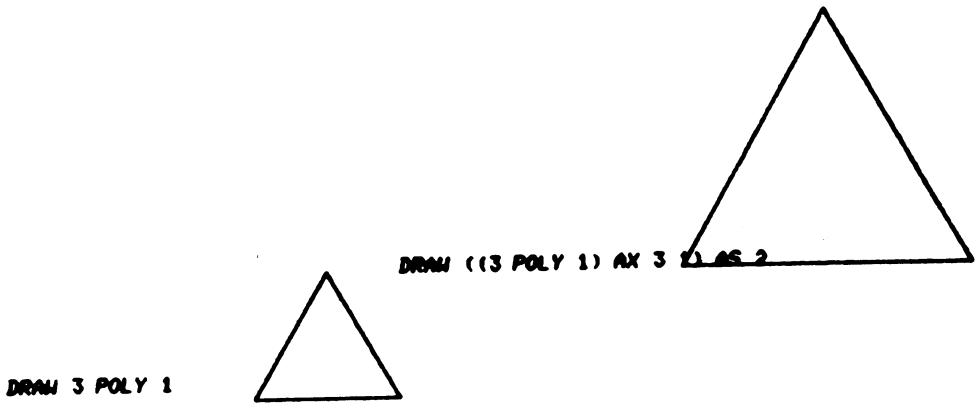
or

DRAW (3 *POLY* 1) *IAT* (*SCALE* 2) *TT XLATE* 3 1 A *ALSO RIGHT*.



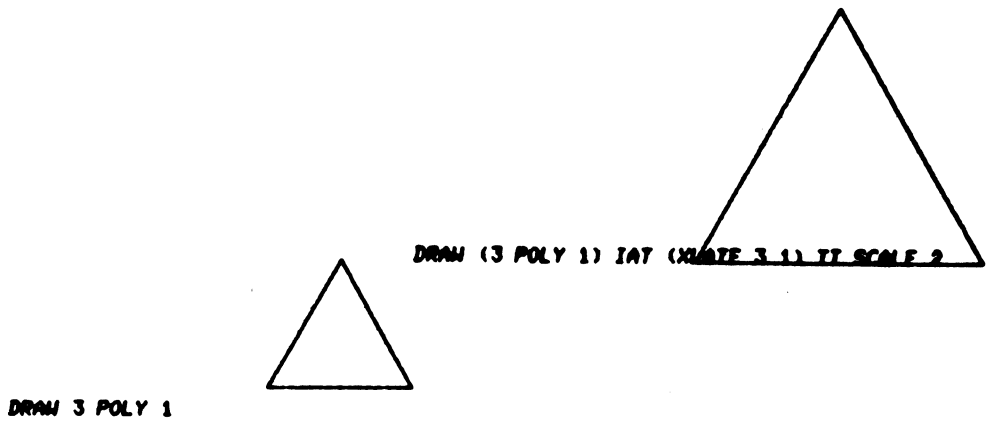
versus

DRAW ((3 *POLY* 1) *AX* 3 1) *AS* 2 ⍝ *WRONG*: *TRANSLATE*, *THEN SCALE*.



or

DRAW (3 *POLY* 1) *IAT* (*XLATE* 3 1) *TT SCALE* 2 ⍝ *ALSO WRONG.*



Scaling about the origin after translation causes the picture to move toward or away from the origin by the scaling factor.

Rotation after translation can be used to create special effects as in the scalloped border in the label (TM) below.

SPECIAL CASES

The following are special transform functions for cases not covered efficiently by combining the three basic transforms:

FUNCTION	SYNTAX	DESCRIPTION
SAP	*TRF←XY SAP SCL*	Scaling transform about point *XY* by scaling factor *SCL*. Equivalent to *TRF←(XLATE -XY) ITT (SCALE SCL) ITT XLATE XY*
RAP	*TRF←XY RAP RU*	Rotation transform about point *XY* for *RU* rotational units (usually degrees).
RSAP	*TRF←(SCL,XY) RSAP RU*	Scaling and rotation transform about point *XY* for *RU* units.
ID	*TRF←ID N*	Identity transform for N-space. Application of an identity transform to a graphic vector or transform, leaves the graphic vector or transform unchanged.
COMIX	*TRF←COMIX N*	Builds comic strip transforms. For example, to set the global variable *CTRANS* to the number of subscreens: *CTRANS←COMIX N* divides the screen into *N*2* subscreens. Each subscreen can be used independently, as if it were the whole screen using the functions. (See **Screen control** in SECTION II-A, Input-Output Functions for examples.)

SUMMARY OF TRANSFORM FUNCTIONS

Scaling	Rotation	Translation	Special
SCALE S	*ROT RU*	*XLATE X*	*(SCL,XY) RSAP RU*
AS S	*AR RU*	*AX X*	*COMIX N*
IAS S	*IAR RU*	*IAX X*	*ID N*
ITS S	*ITR RU*	*ITX X*	
SAP S	*RAP RU*		

SHORT FORMS

1) Transform Applicators

AX	*GV←GV AX XL*	↔ *GV←GV AT XLATE XL*
AS	*GV←GV AS SCL*	↔ *GV←GV AT SCALE SCL*
AR	*GV←GV AR RU*	↔ *GV←GV AT ROT RU*
IAX	*GV←GV IAX XL*	↔ *GV←GV IAT XLATE XL*
IAS	*GV←GV IAS SCL*	↔ *GV←GV IAT SCALE SCL*
IAR	*GV←GV IAR RU*	↔ *GV←GV IAT ROT RU*

2) Transform Combinations

TX	*TRF←TRF TX XL*	↔ *TRF←TRF TT XLATE XL*
TS	*TRF←TRF TS SCL*	↔ *TRF←TRF TT SCALE SCL*
TR	*TRF←TRF TR RU*	↔ *TRF←TRF TT ROT RU*
ITX	*TRF←TRF ITX XL*	↔ *TRF←TRF ITT XLATE XL*
ITS	*TRF←TRF ITS SCL*	↔ *TRF←TRF ITT SCALE SCL*
ITR	*TRF←TRF ITR RU*	↔ *TRF←TRF ITT ROT RU*

SECTION II-C

APL-LIKE FUNCTIONS

(a) **Shape** - the functions SHAPE, ISHAPE and RESHAPE.

FUNCTION: **SHAPE**

SYNTAX: *N←SHAPE GV*

DESCRIPTION: Number of objects in *GV*. ("ρ*GV*" for objects.)

RIGHT PARAMETER *GV*: A graphic vector.

EXAMPLES:

```
      SHAPE 3 4 5 POLY ⍳3
3

      SHAPE 3 4 5 POLY ⍳7
7
```

FUNCTION: **ISHAPE**

SYNTAX: *N←ISHAPE GV*

DESCRIPTION: Number of points in each object in *GV*. ("ρ*GV*" for points.)

RIGHT PARAMETER *GV*: A graphic vector.

EXAMPLES:

```
      ISHAPE 3 4 5 POLY ⍳3
4 5 6

      ISHAPE 3 4 5 POLY ⍳7
4 5 6 4 5 6 4
```

FUNCTION: **RESHAPE**

SYNTAX: *GVR←N RESHAPE GV*

DESCRIPTION: Reshape as in dyadic rho. ("*NρGV*")

LEFT PARAMETER *N*: The number of objects to be produced from *GV*.

RIGHT PARAMETER *GV*: A graphic vector.

EXAMPLES:

```
      SHAPE 3 4 5 POLY ⍳3
3

      SHAPE 7 RESHAPE 3 4 5 POLY ⍳3
7
```

(b) Defining Objects - the functions BEGN and DESC.

FUNCTION: **BEGN**

SYNTAX: *N←BEGN GV*

DESCRIPTION: Origin 1 indices of object delimiters on *GV*.

RIGHT PARAMETER *GV*: A graphic vector.

EXAMPLES:

```
      BEGN 3 4 5 POLY ⍳3
1 5 10
```

FUNCTION: **DESC**

SYNTAX: *N←DESC GV*

DESCRIPTION: Descriptions of objects.

RIGHT PARAMETER *GV*: A graphic vector.

NOTE: *N=(BEGN GV),[1.5]ISHAPE GV*

EXAMPLES:

```
      DESC 3 4 5 POLY ⍳3
 1    4
 5    5
10    6
```

(c) **Index**

FUNCTION: **NDX**

SYNTAX: *GVR←GV NDX N*

DESCRIPTION: Return objects *N* from *GV*. ("*GV*[*N*]")

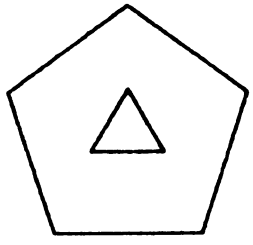
LEFT PARAMETER *GV*: A graphic vector.

RIGHT PARAMETER *N*: Origin 1 indices of the objects to pick from *GV*.

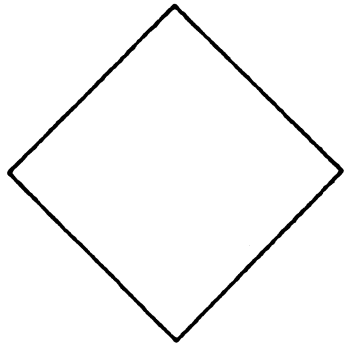
NOTES: $0<(\lceil/N)\leq SHAPE\ GV$

EXAMPLES:

```
DRAW (3 4 5 POLY ⍳3) NDX 1 3
```



```
DRAW (3 4 5 POLY ⍳3) NDX 2
```



FUNCTION: **NDXAS**

SYNTAX: *GVR*←(*GV*1,[1]*GV*2) *NDXAS N*

DESCRIPTION: Index assign. ("*GVR*←*GV*1 ◇ *GVR*[*N*]←*GV*2")

LEFT PARAMETER: *GV*1 and *GV*2 are graphic vectors.

RIGHT PARAMETER *N*: Origin 1 indices of objects in *GV*1 to be replaced by *GV*2.

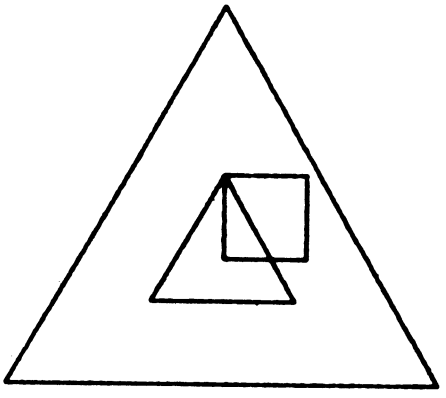
RESULT *GVR*: *GVR* is *GV*1 with objects *N*[*I*] replaced by *GV*2[*I*],0<*I*≤⍴*N*.

NOTE:

(⍴*N*)=*SHAPE GV*2
(*SHAPE GVR*)=*SHAPE GV*1
0<*N*≤*SHAPE GV*1

EXAMPLES:

```
A←3 POLY ⍳3
A←(A,[1]US),NDXAS 2    ⍝US IS A UNIT SQUARE
DRAW A
```



FUNCTION: **NDXG**

SYNTAX: *R←NDXG N*

DESCRIPTION: Index generator. ("⍳*N*")

RIGHT PARAMETER *N*: A two column matrix. The first column is starting indices and the second is the number of indices to be generated.

EXAMPLES:

```
      NDXG 2 2⍴20 5 13 3
20 21 22 23 24 13 14 15


      NDXG 20,[1.5] ⍳4
20 20 21 20 21 22 20 21 22 23
```

(d) i Extended compression

FUNCTION: **OF**

SYNTAX: *R←M OF N*

DESCRIPTION: *M* repetitions of *N*. ("*M*/*N*" - sort of)

NOTES:

```
(⍴M)=1↑⍴N
(1↑⍴R)=+/M
```

EXAMPLES:

```
      0 5 9 OF 1 2 3
2 2 2 2 2 3 3 3 3 3 3 3 3 3

      (⌽⍳3) OF 'ABC'
AAABBC

      2 1 OF 2 2⍴⍳4
1 2
1 2
3 4
```

(d) ii Compression

FUNCTION: **COMP**

SYNTAX: *GVR←B COMP GV*

DESCRIPTION: Compression for graphic vectors. ("*B/GV*")

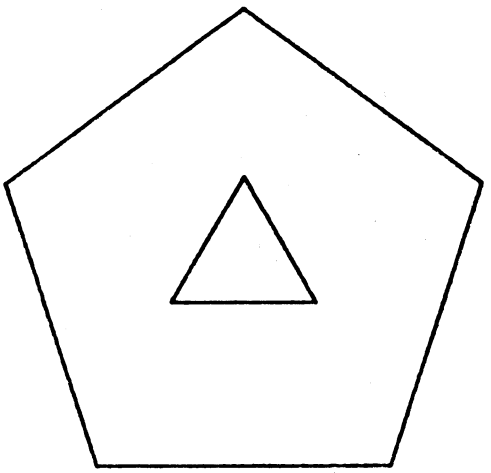
LEFT PARAMETER *B*: Boolean vector.

RIGHT PARAMETER *GV*: Graphic vector to be compressed according to the Boolean vector in the left parameter.

NOTES:
```
(⍴B)=SHAPE GV
(+/B)=SHAPE GVR
```

EXAMPLES:

```
A←3 4 5 6 POLY ⍳4
DRAW 1 0 1 0 COMP A
```



(e) **Make Object Scalar**

FUNCTION: **MOS**

SYNTAX: *GVR←MOS GV*

DESCRIPTION: Removes object delimiters from *GV* - i.e. graphics ravel.

RIGHT PARAMETER *GV*: Graphic vector to be ravelled.

EXAMPLES:

```
      SHAPE 4 5 6 POLY ⍳3
3

      SHAPE MOS 4 5 6 POLY ⍳3
1
```

Partitioned Make Object Scalar

FUNCTION: **PMOS**

SYNTAX: *GVR←B PMOS GV*

DESCRIPTION: Controlled ravel for graphic vectors.

LEFT PARAMETER *B*: Boolean partition vector.

NOTES: *(⍴B)=SHAPE GV*
(+/B)=SHAPE GVR

EXAMPLES:

```
      A←3 4 5 6 POLY ⍳4
      SHAPE A
4
      SHAPE 1 0 1 0 PMOS A
2
```

(f) Take and Drop

FUNCTIONS: **TAKE** and **DROP**

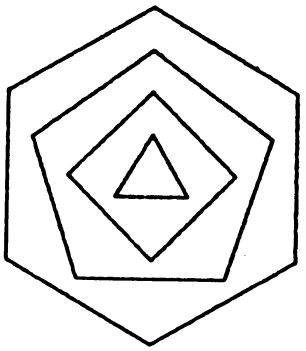
SYNTAX:

```
GVR←N TAKE GV
GVR←N DROP GV
```

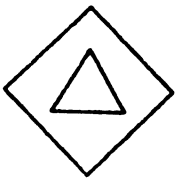
NOTES: TAKE will not overtake.

EXAMPLES:

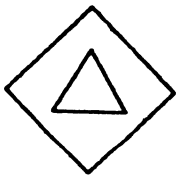
```
A←3 4 5 6 POLY ⍳4
DRAW A
```



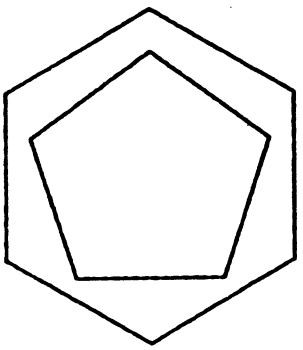
```
DRAW 2 TAKE A
```



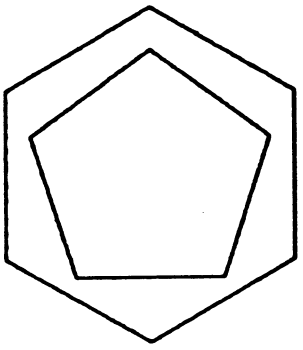
```
DRAW ¯2 DROP A
```



```
DRAW ¯2 TAKE A
```



```
DRAW 2 DROP A
```



(g) Remove Empty Objects

FUNCTION: **REO**

SYNTAX: *GVR←REO GV*

DESCRIPTION: Remove empty objects (objects with no points) from *GV*.

RIGHT PARAMETER *GV*: A graphic vector.

(h) Remove Multiple Moves

FUNCTION: **RMM**

SYNTAX: *GVR←RMM GV*

DESCRIPTION: Multiple moves are removed from *GV*; several consecutive moves are made one move.

RIGHT PARAMETER *GV*: A graphic vector.

SECTION II-D

UTILITY FUNCTIONS

This group contains functions:

ARCTAN, *OF*, *LGRAV*, *PERPD*, *PGRAV*, *MIN*Δ, *AREA*, *PMAXSCAN*, *PPLUSCAN*, *VTOM*, *MTOV* and *PTGRD*.

(a) Point Grids

FUNCTION: **PTGRD**

SYNTAX: *R*←*PTGRD N*

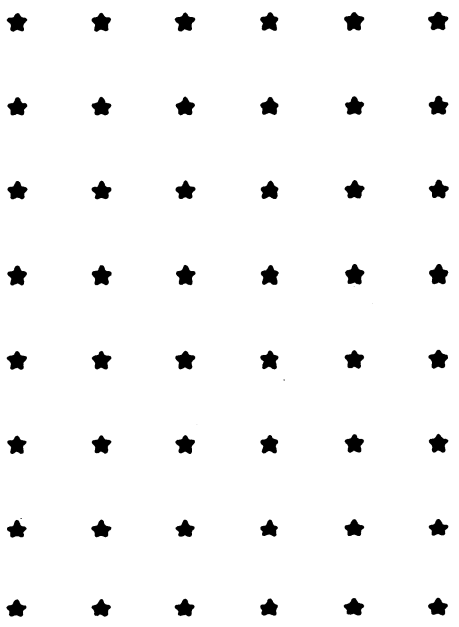
DESCRIPTION: Generates translation coordinates for point grids, suitable for *AX* or *IAX*.

RIGHT PARAMETER:

N[1 2]= grid size in elements X by Y
N[3 4]= interelement spacing X,Y
(if omitted, 1 1 is assumed)
N[5 6]= lower left corner of the grid
(if omitted, 0 0 is assumed).

EXAMPLES:

```
DRAW (5 5 DAISY .1) IAX PTGRD 6 8 1 1 ¯3 ¯4
```



(b) **Arctan**

FUNCTION: **ARCTAN**

SYNTAX: *R←ARCTAN XY*

DESCRIPTION: Returns the arctan of the angle between point *XY* and the X axis.

RIGHT PARAMETER *XY*: A two-column matrix of cartesian coordinates (*X*,*Y*).

(c) **Area**

FUNCTION: **AREA**

SYNTAX: *R←AREA GV*

DESCRIPTION: The area enclosed by *GV*. Positive if traced clockwise; negative if traced counterclockwise. *GV* should be a closed object.

EXAMPLES:

```
 AREA 4 POLY 2*.5
2
```

(d) **Arcs**

FUNCTION: **CIRCLE**

SYNTAX: *GV←CIRCLE XY*

DESCRIPTION: Defines circular arc through the three points given in *XY*. 3 2 = ρ*XY*.

(e) **Line Gravity**

FUNCTION: **LGRAV**

SYNTAX: *GVR←XY LGRAV GV*

DESCRIPTION: Drops a perpendicular line from a point *XY* to the **nearest** point on *GV*.

(f) Minimum Distance

FUNCTION: **MIN∆**

SYNTAX: *N1←XY MIN∆ N*

DESCRIPTION: Returns the point from list *N* (calculated by *PERPD*) that is closest to *XY*.

(g) Perpendiculars

FUNCTION: **PERPD**

SYNTAX: *N←XY PERPD GV*

DESCRIPTION: Drops perpendicular lines from a point *XY* to all lines in an object.

(h) Point Gravity

FUNCTION: **PGRAV**

SYNTAX: *N1←XY PGRAV GV*

DESCRIPTION: Returns the point on *GV* closest to *XY*.

(i) Partitioned Max Scan

FUNCTION: **PMAXSCAN**

SYNTAX: *R←B PMAXSCAN N*

LEFT PARAMETER: *B* is a Boolean partition vector.

RIGHT PARAMETER: *N* is a numeric vector.

EXAMPLES:

```
      1 0 0 1 0 0 0 PMAXSCAN 8 7 9 5 3 4 12
8 8 9 5 5 5 12

      ⌈\8 7 9 5 3 4 12
8 8 9 9 9 9 12
```

(j) Partitioned Plus Scan

FUNCTION: **PPLUSCAN**

SYNTAX: *R←B PPLUSCAN N*

LEFT PARAMETER: Boolean partition vector.

RIGHT PARAMETER: N is a numeric vector.

EXAMPLES:

```
      1 0 0 1 0 0 0 PPLUSCAN 7⍴1
1 2 3 1 2 3 4

     +\7⍴1
1 2 3 4 5 6 7
```

(k) Vector To Matrix

FUNCTION: **VTOM**

SYNTAX: *M←VTOM CH*

DESCRIPTION: *VTOM* produces a character matrix from *CH*.

RIGHT PARAMETER: *CH* is a character vector.
1↑*CH* delimits substrings within *CH*.

EXAMPLES:

```
      VTOM '/A/BCD/EF/G/H'
/A
/BCD
/EF
/G
/H
```

(l) **Matrix To Vector**

FUNCTION: **MTOV**

SYNTAX: *V←CH MTOV M*

LEFT PARAMETER *CH*: Trailing *CH* are removed.

RIGHT PARAMETER: Matrix to be converted to a character string. The first column of the matrix *M* should be delimiters.

EXAMPLES:

```
      M←VTOM '/A/BCD/EF/G/H'
      ' ' MTOV M
/A/BCD/EF/G/H
```

SECTION II-E

THREE-SPACE FUNCTIONS

When working in three dimensions, the 3-dimensional object created has to be converted to a two-dimensional projection of itself before it can be drawn. The viewpoint is the screen origin (0,0,0) and viewing is done outward along the positive Z-axis. The picture plane is perpendicular to the Z-axis. The distance between the viewer and the picture plane determines the size of the picture. Objects closer than the picture plane will be badly distorted.

Wire-frame drawing with back plane removal is supported at present. Hidden line algorithms do exist for these workspaces, but are very expensive. Single point (camera) perspective is used. The user must do the 3-D clipping.

The functions in this group are:
TO2D, *VIEW*, *POLY3*, and *SOLID* and *CUBE*.

Execute the function *THREED* to define these functions in the workspace.

(a) View

FUNCTION:	**VIEW**
SYNTAX:	*GV←N VIEW GV3D*
LEFT PARAMETER:	*N*[1] is the distance of the picture plane from the viewer. It is effectively a scalar magnification applied to the result. *N*[2] is a translation to be applied to the 3-D objects *GV3D* along the Z-axis, before viewing them.
RIGHT PARAMETER:	Must a be three-dimensional graphics vector, (i.e. there must be 4 columns in the matrix *GV3D*) .
RESULT *GV* :	The two-dimensional view of the three-dimensional objects in *GV3D* with back plane removal.
NOTE:	The statement *DVIEW GV3D* is equivalent to *DRAW VPT VIEW GV3D*. *VPT* has the default value of 50 120.

(b) **To Two-Dimensional View.**

FUNCTION: **TO2D**

SYNTAX: *GV2D←N TO2D GV3D*

DESCRIPTION: TO2D is similar to VIEW without back plane removal.

PARAMETERS: The parameters are the same as for VIEW.

NOTE: The statement *DTO2D GV3D* is equivalent to *DRAW VPT TO2D GV3D*.
VPT has the default value of 50 120.

(c) **Cylinders**

FUNCTION: **POLY3**

SYNTAX: *GV←N1 POLY3 N2*

DESCRIPTION: Produces the end-planes of a three-dimensional N-gon cylinder.

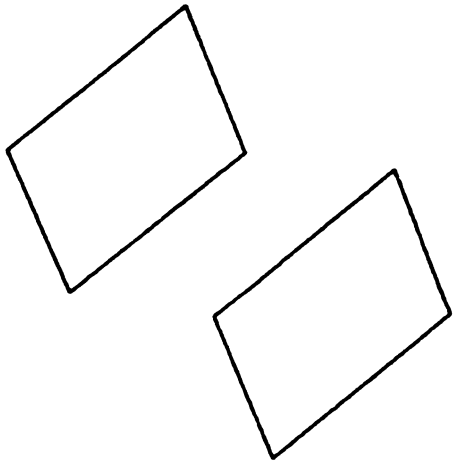
LEFT PARAMETER: *N1* is the number of sides.

RIGHT PARAMETER: *N2* is the radius of each polygon in *N1*.

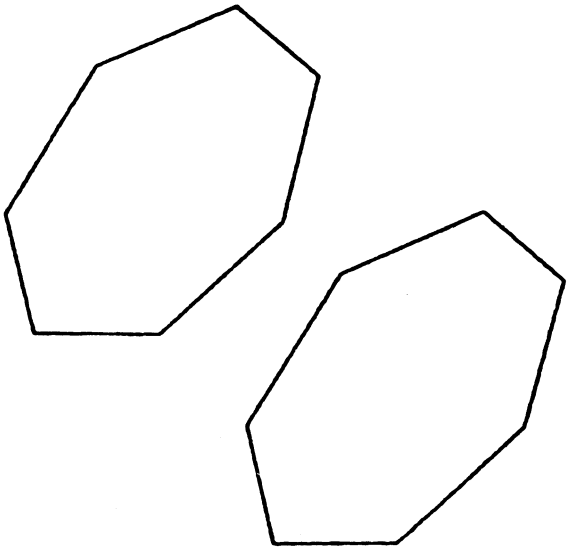
NOTE: The length of the cylinders is twice the radius.

EXAMPLES:

```
DRAW 50 120 TO2D (4 POLY3 1) AT ROT 1 3⍴30 45 0
```



```
DRAW 50 120 VIEW (7 POLY3 1) AT ROT 1 3ρ30 45 0
```



(d) Wire Frame Drawings

FUNCTION: **SOLID**

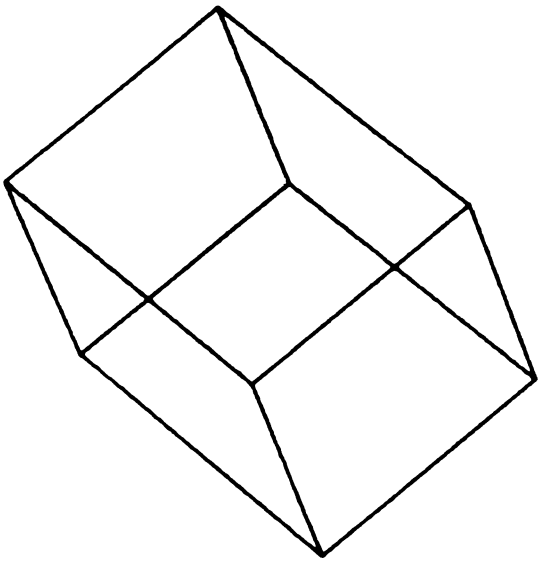
SYNTAX: *GVR←SOLID GV*

DESCRIPTION: Produces a wire frame drawing of a cylinder from the end planes supplied by *POLY*3.

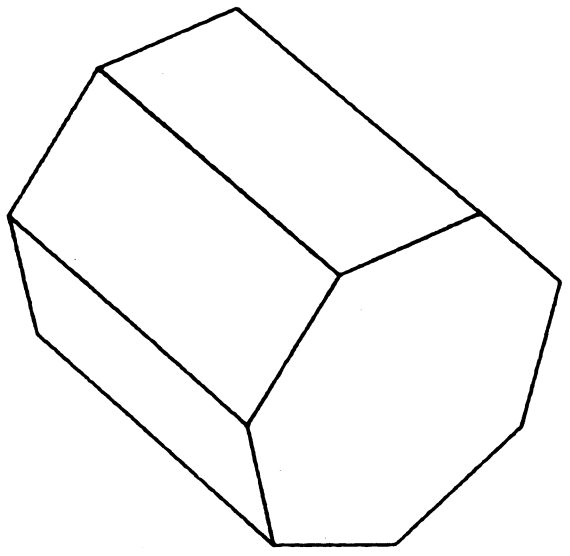
RIGHT PARAMETER *GV*: Graphics vector containing the end planes of a cylinder.

EXAMPLES:

```
DRAW 50 120 TO2D SOLID (4 POLY3 1) AT ROT 1 3p30 45 0
```



DRAW 50 120 *VIEW SOLID* (7 *POLY*3 1) *AT ROT* 1 3ρ30 45 0



(e) **Cubes**

FUNCTION: **CUBE**

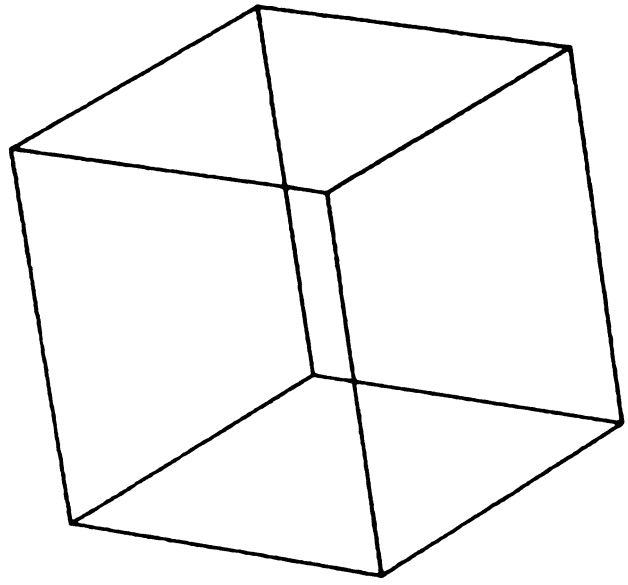
SYNTAX: *GV3D←CUBE N*

DESCRIPTION: Produces an exact cube with sides of length *N*.

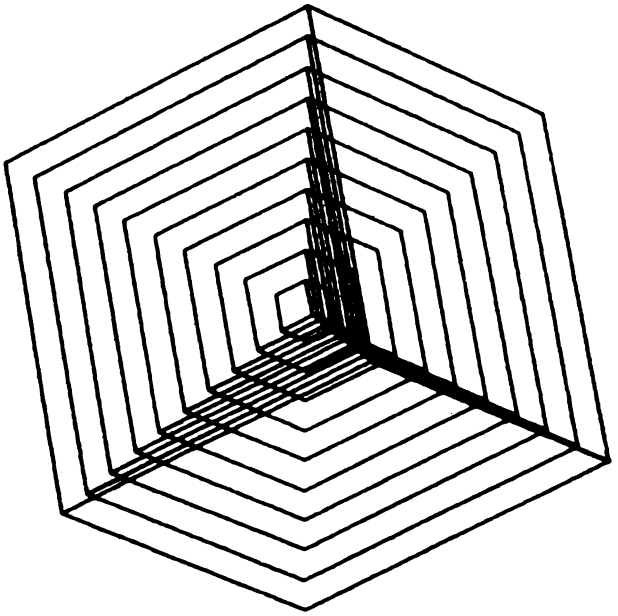
RIGHT PARAMETER *N*: Radius of cube *GV3D*.

EXAMPLES:

```
DRAW 50 120 TO2D SOLID (CUBE 1) IAR 1 3⍴30 45
```



```
DRAW 50 120 VIEW SOLID (CUBE .1×⍳10) IAR 1 3⍴30
```



SECTION II-F

FILE FUNCTIONS

The following functions simplify the storage and retrieval of objects on file.

FUNCTION	SYNTAX	COMMENTS
FIND	*F←LIB FIND NAME*	Look up *NAME* in the directories of the files *LIB* (one file name per row). *F* is the file tie number, component numbers that the objects in *NAME* occupy.
GET	*GV←GET NAME*	Look up and read the first component of *NAME* from the file library. The global variable *LIB* contains the id's of files to be searched for *NAME*.
QT	*N←QT FNAME*	The result is the tie number of *FNAME*. If the file is not tied, *QT* □*STIE*'s it.
QCRE8	*N←QCRE8 FNAME*	Similar to *QT* but creates *FNAME* if it does not exist.
FSHAPE	*N←FSHAPE F*	File shape. i.e. the number of objects.
FISHAPE	*N←FISHAPE F*	File itemwise shape. i.e. the lengths of objects.
FBLOCK	*N←FBLOCK F*	Data is reblocked to approximately *N* points per component. Each component is a minimum integral number of objects, such that (1↑⍴*COMPONENT*) ≥ *N*.
FIAT	*F←F FIAT TRF*	File itemwise apply transform.
FAT	*F←F FAT TRF*	File outerproductly apply transform.
DRAW	*DRAW F*	Draw - see Input/Output functions.
TABTOFILE	*F←N TABTOFILE FNAME*	Graphic tablet to file. *N* is the same as in the graphic tablet function *TABLET*.

APPENDIX A

GLOSSARY

Abbreviations

GV	a graphic vector i.e. zero or more objects
TRF	zero or more transforms
XL	translation amounts
SCL	scaling amounts
RU	rotational units
N	numeric data
CH	character data
SV	text string vector (substrings delimited by 1↑*SV*)
XY	cartesian data
B	Boolean partition vector, (ones start each partition); or Boolean compression/expansion vector
F	file tie number, component-number vector
FNAME	file name
BX	zero or more boxes.

Boxes

Boxes may be used for placement of text, or for moving pictures about in a manner akin to "viewport/window" techniques.

A box is an Nx4 matrix representing a collection of N rectangles. In standard format, each row of the matrix represents the cartesian (X,Y) coordinates of the lower left and the upper right corners of the box.

Global Variables

TRANS

TRANS is a transform which maps the arbitrary 2-space of the user into a unit-radius origin centered space. This allows the user to change the portion of the universe displayed on the screen. The default setting is *TRANS*←*SCALE*÷10 i.e. radius 10, origin centered.

CTRANS

CTRANS is an array of transforms which may be used to divide the screen into subscreens. Each plane of *CTRANS* represents one subscreen. All subscreens have the same aspect ratio as the whole screen. The *NEXTFR* function causes *DRAW* to use the next subscreen. When the last frame is used, a *WAITCR* ◇ *ERASE* is executed. *FRAME N* causes the *DRAW* function to use *FRAME N*, and *WHATFR* returns the current frame number.

SMTRANS

SMTRANS is a terminal dependent transform that maps the unit radius origin centered clipping space into the physical terminal screen space. Most users will never need to modify *SMTRANS*. This transform is defined by the *TERM* function.

SV

SV is the state variable and contains assorted data used by the graphics system. Selected elements are documented in the file *SVDOC*.

OD

OD is a vector of controls defining a set of object delimiters. Its default is ¯4 ¯2.

PD

PD is the polygon delimiter, and is used by certain three-dimensional space functions. Its default is ¯8 ¯6.

DEGR

A multiplicative conversion factor from the user's measure of rotation (usually degrees) to radians which are used by the APL circle functions. This allows the user to specify rotation in the most convenient units for a particular application. The conversion is performed automatically by the graphics functions.
If *DEGR* is positive, then positive rotations are clockwise and negative rotations counterclockwise. If *DEGR* is negative, the opposite is true.
The default value of *DEGR* is (○1)÷180, the conversion between degrees and radians. If you plan to use rotational units other than degrees, *DEGR* should be set accordingly.

Objects, Graphic Vectors and Controls

A collection of points and/or line segments in space is called an **object**. It is represented as a set of controls (pen up; pen down; draw a point; new object) and cartesian coordinates defining a line drawing (X,Y when working in a single plane i.e. in two dimensions or 2-space; X,Y,Z when working in three dimensions or 3-space). The first control on an object must be an object delimiter.

An object is represented as a numeric matrix (*GV*). Each row of the matrix represents a point of the object; the columns are assigned as follows:

```
(N,3) = ρGV --> 2-space
(N,4) = ρGV --> 3-space
(N,M) = ρGV --> (M-1)-space
```

```
GV[;1] = Controls (Beginning of object, move, draw, make point, pen type)
GV[;2] = First coordinate (X)
GV[;3] = Second coordinate (Y)
GV[;M] = (M-1)th coordinate (Z....)
```

An object is treated as an entity by the graphics system in much the same manner as a scalar is treated by APL. An object may be thought of as a graphic scalar and most graphic functions treat an object as a scalar or a one-element vector. Zero or more objects, catenated along the first coordinate, form a picture or **graphic vector**. Most graphic functions treat an N-object picture as an N-element vector.

Controls

Individual objects within a graphic vector are delimited by a control called the **object delimiter**. An object delimiter (¯4 is the default value) indicates the beginning of an object. The global variable *OD* defines a set of controls that are object delimiters. Recognized controls include:

```
¯4=CONTROL          --> Object delimiter
¯2=CONTROL          --> Pointplot object delimiter
¯8=CONTROL          --> Polygon delimiter
¯6=CONTROL          --> Pointplot polygon delimiter
```

Other controls are:

```
0=2|CONTROL         --> Move to XY (pen up)
1=2|CONTROL         --> Draw to XY (pen down)
1↑0 100⊤ CONTROL    --> Pen type on certain terminals (Tektronix 4015)
2=4|CONTROL         --> Point plot (move to XY and draw point)
```

Each object in a graphic vector may be treated independently of the other objects, or the graphic vector may be manipulated as a whole.

A limited set of APL-like functions is provided to manipulate graphic vectors, yielding capabilities such as *SHAPE*, *RESHAPE*, *TAKE*, *DROP*, *COMPRESS* and *INDEX*. (Group *APL* in workspace 3 *GRAPHICS*.)

Objects on Files

Objects may be stored on files and a small set of functions is provided to manipulate such objects (the group *FILES* in workspace 3 *GRAPHICS*). Objects on files are represented by a vector consisting of a file tie number, followed by a vector of component numbers within that file. Most file functions append their output to the file named by the variable *WORKFILE*.

Text - A Vector of Character Strings

Text is represented as a vector of character strings (*SV*). Strings are vectors of characters or numbers, delimited by their first element. This allows a limited implementation of non-recursive deep generalized arrays.

Analogous to graphic vectors, each character string (text object) is delimited by a special character - specifically the first character of the text vector.

Care must be taken not to use a delimiter as part of a character string. The character following the string delimiter specifies the type font to be used with that string. The numbers 0 to 7 are hardware characters specific to the terminal in use. In most cases they are the standard APL characters. Fonts greater than 7 are the more commonly used APL characters supported as stroke (drawn) characters.

Terminals

Most graphic input functions are terminal dependent. Input functions are provided for Tektronix 4000 and Hewlett/Packard 2648A series crosshair input "Gin mode", and the Tektronix 4953 graphic tablet. Most non-CRT terminals do not support graphic input at all.

The following terminal types are supported (today). To condition the workspace, type: *TERM* **'code'**, where **code** is one of the codes given below. For example, when using any Tektronix 4000 series terminal, the first commands after signing on would be:

```
)LOAD 3 GRAPHICS
TERM 'TEK'
```

Terminal	Code
Any Tektronix 4000 series terminal	*TEK*
Tektronix 4013	*TEK*4013
Tektronix4015	*TEK*4015
Anderson-Jacobson AJ832	*AJ*832
Xerox Diablo model 1550	*DIABLO*
Xerox Diablo 1620	*DIABLO*
Xerox Diablo 1700	*NEWD*1700
DTC300	*DTC*300
DTC302	*DTC*302
Trendata 4000	*TREN*4000
Trendata 4000A	*TREN*4000*A*
Multiwriter II	*ASL*2
Multiwriter III	*ASL*3
Tektronix 4662 Plotter	*T*4662
Houston Instruments Plotter	*HIPPO*
Hewlett/Packard 7221A	*HP*7221
Hewlett/Packard 9221A	*HP*9221
Spintronic	*SPIN*2
Mitek	*MITEKV*3

To obtain an updated list of supported terminals and their codes, type *TERM* **''**.

Terminal-specific strapping and setup requirements are documented in the *OUTTERM* function, created by successful execution of the *TERM* function.

Supporting new terminal types is rather easy, as there is only one terminal-dependent output function (called *OUTTERM*) in the workspace. Thus, one can run the same set of functions on a different terminal by executing the *TERM* function to define the *OUTTERM* function necessary for the terminal in use.

The illustrations in this manual were produced on a Tektronix 4015 with a 4631 hardcopy attachment.

Transforms

Transforms (*TRF*) are rank-3 arrays that describe the movement and position of objects in space. Each plane represents a single linear transform that may be applied to a single object or to all objects in a graphic vector by use of the dyadic functions *IAT* and *AT* respectively.

For pictures in two dimensions, a single transform is a 1x3x3 array representing values of rotation, translation and scaling, while for three-dimensional objects it is a 1x4x4 array. Transforms may generally be combined by catenation along the first coordinate. Most graphic functions treat a transform as a one-element vector or scalar.

BASIC TRANSFORMS

Scaling: The process of shrinking or magnifying an object, and may be considered analogous to "zoom". Scaling may take place about the coordinate origin (*SCALE* or *IAS*), or about a specified point (*SAP*).

Rotation: The process of rotating an object about a point, either the coordinate origin (*ROT* or *IAR*), or about a specified point (*RAP*).

The function *RSAP* may be used to simultaneously rotate and scale about a point.

Translation: The process of moving objects. The functions *XLATE* or *IAX* applied to an object will move it in a manner analogous to "re-position", "tilt" and "pan".

TRANSFORM GENERATORS

Transform generating functions, including *SCALE*, *ROT*, *XLATE* and others (APPENDIX B page 129) contains complete lists of functions) generate arrays that will scale, rotate or translate the object when the transform is applied.

For example, *XLATE X* (where *X* is a scalar), will produce a transform (a 1x3x3 matrix) that can be applied to an object to move it *X* units to the right if *X*>0 and to the left if *X*<0.

SCALE S will produce a transform that will enlarge an object by *S* units if *S*>1, or scale down by *S* if 0≤*S*<1. *COMIX N* builds transforms to divide the screen into *N**2 subscreens when assigned to the global variable *CTRANS*.

TRANSFORM APPLICATORS

Once transforms have been generated, we need functions to **apply** them to graphic vectors. The functions are *AT* and *IAT*. *AT* will apply each transform to all the objects, while *IAT* will apply transforms to each object in the graphic vector one by one (itemwise).

COMBINING TRANSFORMS

Transforms may be combined when two or more must be applied to the same graphic vector. The first transform is transformed by the second, giving a combined transform to be applied to the graphic vector. The functions are *TT* (every transform on the right is applied to every transform on the left, as in outer-product) and *ITT* (itemwise) combine transforms.

When transforms are combined they act as if they had been applied separately in the order from left to right. The order in which transforms are combined is important, and should be

SCALE-->ROTATE-->TRANSLATE.

APPENDIX B

REFERENCE CARD

INPUT-OUTPUT FUNCTIONS & OBJECT GENERATORS

Name	Syntax	Description	Page T# is text/E# example
BOX	*GV←BOX BX*	Generates boxes *BX*.	T5,20,57,60/E60,62
CLIP	*GVR←BX CLIP GV*	Clip *GV* to box *BX*.	T63/E63,64
DAISY	*GV←N1 DAISY N2*	Generates daisies. - *N1*[1] is the number of points to draw. - *N1*[2] is the number of petals. - *N2* is the radii.	T13-15,50/E13-15,39,51...
DASH	*GV←N DASH GV*	Dashed-line *GV*. *N* is a vector of dash lengths and interdash spaces.	T74/E74
DRAW	*DRAW GV*	Graphic output. - (*N*,3)=ρ*GV* : *GV* is clipped to the screen boundary and drawn; - (1=ρρ*GV*)∧1<ρ*GV* : □*READ*, clip and draw data from file. 1↑*GV* is file tie number, 1↓*GV* are components to be drawn; - (1=ρ,*GV*) : □*READ*, clip and draw data from entire file. *GV* is the file tie number.	T19,55/E everywhere
DRAWINBT	*N DRAWINBT GV*	Same as: *DRAW N INBT GV*	T73/E73

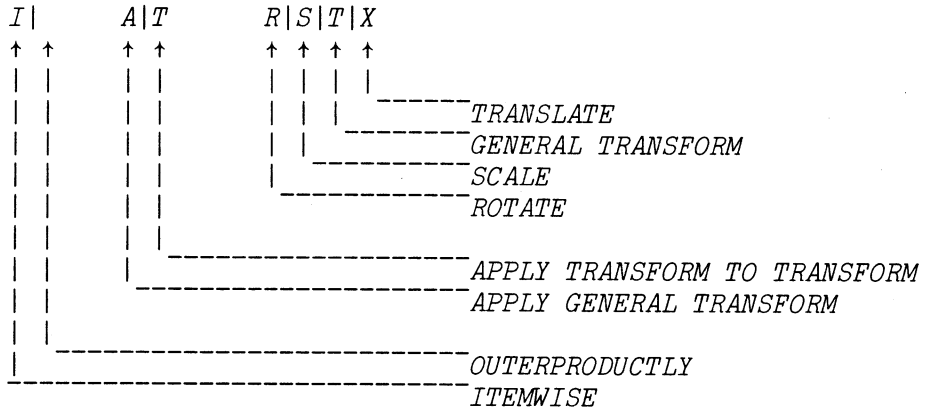
DRAWSTR	*BX DRAWSTR SV*	Stroke characters: Draws stroke characters inside the boxes specified. Type fonts: 0,1,2,3,4,5,6,7 are hardware characters; 8,9 are stroke characters.	T20,57/E20,57
ERASE	*ERASE*	Erase screen and home cursor (form feed on Diablos). Useless on Multiwriter II.	T70
FOLLOW	*GV←FOLLOW*	Interactive crosshair input. Draws as it goes. Input characters are: Numerics used directly as controls; N - New object P - Draw to nearest point L - Draw to nearest line C - Circular arc through last, current and next points.	T17,53/E17,42,53
FRAME	*FRAME N*	Screen control. Causes frame (1↑⍴*CTRANS*)|*N* to be used for the next draw.	T70/E70

GRID	*GV←GRID N*	Line grid. - *N*[1 2] = grid size in elements, - *N*[3 4] = interelement spacing, default is 1 1, - *N*[5 6] = lower left corner, default is 0 0	T66/E66
HATCH	*GVR←N HATCH GV*	Cross-hatch *GV*. *N*[1] = Angle of lines *N*[2] = Crosshatch spacing.	T75/E38,75
IMM	*BX←IMM GV*	Itemwise minimum/maximum Generates boxes that fit exactly around objects in *GV*.	T65/E65-66
INBOX	*GVR←GV INBOX BX*	Scale *GV* to fit inside *BX*.	T61/E61-62
INBT	*N INBT GV*1,[1]*GV*2	In between. - (1=×/ρ*N*) Interpolate *N*-1 objects between *GV*1 and *GV*2; - (1<×/ρ*N*) Interpolate 1+ρ*N* objects relatively spaced by *N*.	T73/E73
MENU	*N←MENU SV*	Select from the menu *SV* with crosshairs. *N* gets string indices.	T67/E68

MO	*GV←MO N*	Make object from vector.	T10,48/E10,48
NEXTFR	*NEXTFR*	Next frame. Switch to drawing in the next frame. *WAITCR ◇ ERASE* when frames are used up.	T71/E71
PEN	*GVR←GV1 PEN N*	Alter controls to pen type N.	
POLY	*GVR←N1 POLY N2*	Generates polygons. *N1*-sided polygons of radii *N2*.	T11,49/E11-12,22+,34,37,49,etc...
STR	*GV←STR CH*	Stroke characters. One object is produced per character.	T59/E59
STR5	*GV←STR5 SV*	Stroke characters. One object is produced per string.	T59/E59
SQPLOT	*GV←SQPLOT N*	Square plot of vector *N*.	T72/E72
TABLET	*GV←TABLET N*	Graphic input tablet: Tektronix 4953. *N*=0- point to point *N*=1- continuous (curve tracing).	T18,54/E18,54
TEXT	*XY TEXT SV*	Hardware characters. Hardware character text output at locations X, Y. 1ρ*CH* is the character string delimiter, first character after each delimiter is the character type.	T58,/E33,58
VTEXT	*GV←VTEXT SV*	Vertical text. Inserts linefeeds and backspaces.	T58/E58
WAITCR	*WAITCR*	Wait for carriage return.	T71
WHATFR	*N←WHATFR*	Current frame number.	T72
XIN	*GV←CH XIN N*	Crosshair input. *N* is the maximum number of points; 0 is no limit. *CH* are acceptable input characters.	T16,52/E16,46,52

The following diagram will help to explain some of the naming conventions adopted for transforms.

Nouns:	R	Rotation
	S	Scaling
	T	General transform (TRF)
	X	Translation
Verbs:	A	Apply general transform
	T	Apply transform to transform
Adverbs: (blank)		Apply all transforms to all objects (outer-product)
	I	Itemwise apply transform (object by object)



TRANSFORM FUNCTIONS

Name	Syntax		Description	Page T# is text / E# example
Transform Generators				
SCALE	*TRF←SCALE SCL*		Scale.	T78,79/E78
ROT	*TRF←ROT RU*		Rotate.	T78,80/E40,41,42,84-87
XLATE	*TRF←XLATE XL*		Translate.	T78,81/E30,31,41,84-87
SAP	*TRF←XY SAP SCL*		Scale about point.	T78,88
RAP	*TRF←RAP RU*		Rotate about point.	T78,88
RSAP	*TRF←(SCL,XY) RSAP RU*		Rotate and scale about point.	T78,88,79
ID	*TRF←ID N*		N-space identity transform.	T88
COMIX	*TRF←COMIX N*		NxN subscreen transform.	T69,88 E69
Combining Transforms				T83/E84-87
TT	*TRFR←TRFA TT TRFB*		Transform transform.	
ITT	*TRFR←TRFA ITT TRFB*		Itemwise transform transform.	
Short Forms for Transform Combinations				T89
TX	*TRF←TRF TX XL*	=	*TRF←TRF TT XLATE XL*	
TS	*TRF←TRF TS SCL*	=	*TRF←TRF TT SCALE SCL*	
TR	*TRF←TRF TR RU*	=	*TRF←TRF TT ROT RU*	
ITX	*TRF←TRF ITX XL*	=	*TRF←TRF ITT XLATE XL*	
ITS	*TRF←TRF ITS SCL*	=	*TRF←TRF ITT SCALE SCL*	
ITR	*TRF←TRF ITR RU*	=	*TRF←TRF ITT ROT RU*	
Transform Applicators				T82/E84-87
AT	*GVR←GV AT TRF*		Outerproductly apply transform.	
IAT	*GVR←GV IAT TRF*		Itemwise apply transform.	
Short Forms for transform generators and applicators				T89
AX	*GV←GV AX XL*	=	*GV←GV AT XLATE XL*	
AS	*GV←GV AS SCL*	=	*GV←GV AT SCALE SCL*	
AR	*GV←GV AR RU*	=	*GV←GV AT ROT RU*	
IAX	*GV←GV IAX XL*	=	*GV←GV IAT XLATE XL*	
IAS	*GV←GV IAS SCL*	=	*GV←GV IAT SCALE SCL*	
IAR	*GV←GV IAR RU*	=	*GV←GV IAT ROT RU*	

APL LIKE FUNCTIONS

Name	Syntax	Description	Page T# is text/E# example
BEGN	*N←BEGN GV*	Beginning of objects.	T93/E93
COMP	*GVR←B COMP GV*	Compression.	T97/E27,97
DESC	*N←DESC GV*	Description of objects.	T93,/E93
DROP	*GVR←N DROP GV*	Drop.	T99/E26,99-100
ISHAPE	*N←ISHAPE GV*	Itemwise shape: i.e. the lengths of objects.	T91/E34,91
MOS	*GVR←MOS GV*	Make object scalar.	T98/E29,98
NDX	*GVR←GV1 NDX N*	Index.	T94/E94
NDXAS	*GVR←N NDXAS GV1*`,[1]`*GV2*	Index assign. `(⍴`*N*`)=`*SHAPE GV2* *GVR←GV1* ◇ *GVR*`[`*N*`]←`*GV2*	T95/E95
NDXG	*R←NDXG N*	Index generator.	T96/E34,96
OF	*GVR←N OF GV*	*N* repetitions of *GV*. Integer left argument extension of compress.	T96/E96
PMOS	*GVR←B PMOS GV*	Partitioned *MOS*. *B* is a Boolean partition vector and `(+/`*B*`)↔`*SHAPE GVR*	T98/E28,98
RESHAPE	*GVR←N RESHAPE GV*	Reshape.	T92/E92
REO	*GVR←REO GV*	Remove empty objects.	T100/E100
RMM	*GVR←RMM GV*	Remove multiple moves.	T100
SHAPE	*N←SHAPE GV*	Shape of *GV*.	T56,91/E91
TAKE	*GVR←N TAKE GV*	Take, not overtake.	T99/E24,25,28,29,99,100

UTILITY FUNCTIONS

Name	Syntax	Description	Page T# is text/E# example
AREA	*N←AREA GV*	Itemwise signed area.	T102
ARCTAN	*N←ARCTAN XY*	Arctan. *N* is true arctan for XY in radians.	T102
CIRCLE	*GV←CIRCLE XY*	Circular arc. through 3 point in *XY*.	T102
LGRAV	*GVR←XY LGRAV GV*	Drop perpendicular line.	T102
MIN∆	*N1←XY MIN∆ N*	Nearest point.	T103
MTOV	*SV←CH MTOV M*	Matrix to string vector. Trailing *CH* are removed. First column of matrix *M* should be delimiters.	T105/E105
PERPD	*N←XY PERPD GV*	Perpendiculars to all lines.	T103
PGRAV	*N1←XY PGRAV GV*	Closest point on *GV*.	T103
PMAXSCAN	*N←B PMAXSCAN N*	Partitioned max scan.	T103/E103
PPLUSCAN	*N←B PPLUSCAN N*	Partitioned plus scan.	T104/E104
PTGRD	*XY←PTGRD N*	Translation amounts for grid *N*.	T101/E101
VTOM	*M←VTOM CH*	Matrix to vector.	T104/E104

THREE SPACE FUNCTIONS

Name	**Syntax**	**Description**	**Page** T# is text/E# example
CUBE	*R←CUBE N*	Exact cube.	T112/E112
POLY3	*GV←N1 POLY3 N2*	Three-space polygonal cylinders.	T108/E109,110,111,113
SOLID	*GVR←SOLID GV*	Polyhedra bounded by pairs of objects from *GV*	T110/E110,112,113
TO2D	*GVR←N TO2D GV*	Convert to 2-space; - *N*[1]= distance of viewer from origin, - *N*[2]= distance from viewer to picture plane.	T108/E108,110,112
VIEW	*GVR←N VIEW GV*	*TO2D* with back plane removal.	T107/E109,111,113
VISIBLE	*B←VISIBLE GV*	Visibility of objects in *GV*. Clockwise traced objects are visible.	

FILE FUNCTIONS

Name	Syntax	Description	Page
			115
DRAW	*DRAW F*	Draw - see input-output functions.	
FAT	*F←F FAT TRF*	File outerproductly apply transform.	
FBLOCK	*F←N FBLOCK F*	Data is reblocked to approximately N points per component. Each component is a minimum integral number of objects, such that (1↑⍴*COMPONENT*)≥*N*	
FIAT	*F←F FIAT TRF*	File itemwise apply transform.	
GET	*GV←GET NAME*	Find *NAME* and read the first component.	
FIND	*F←FIND NAME*	Find *NAME* in the directory of the file library.	
FISHAPE	*N←FISHAPE F*	File itemwise shape, i.e. lengths of objects.	
FSHAPE	*N←FSHAPE F*	File shape, i.e. number of objects.	
TABTOFILE	*N TABTOFILE FNAME*	Graphic tablet to file. N is the same as in the *TABLET* function.	
QCRE8	*N←QCRE8 FNAME*	Similar to *QT* but creates *FNAME* if it doesn't exist.	
QT	*N←QT FNAME*	*N* gets tie number of *FNAME*. If not tied, ⎕*STIE FNAME*.	

GLOBAL VARIABLES

Name	Description	118
CTRANS	*COMIX* and subscreen transforms.	
DEGR	Conversion factor from degrees to radians.	
OD	Object delimiter.	
PD	Polygon delimiter.	
SMTRANS	Terminal-dependent transform.	
SV	Terminal state vector and various constants.	
TRANS	User space to unit radius, origin centred transform.	